MASTER EN SCIENCES INFORMATIQUES

Un manipulateur de bloc de données

Zune, J.M.

Award date:
1981

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

UN MANIPULATEUR DE BLOC DE DONNEES.

PAR J.M. ZUNE

Qu'il me soit permis d'exprimer toute ma reconnaissance aux Professeurs J. HENRARD et Cl. CHERTON, promoteurs de ce mémoire, pour la collaboration et l'aide bienveillante dont ils m'ont fait bénéficier.

Je tiens à remercier tout particulièrement Messieurs Ph. DONTAINE et H. CLAES pour l'extrême disponibilité dont ils ont fait preuve à mon égard.

J'adresse enfin tous mes remerciements à Messieurs D. STANDAERT et J.-M. BOUDRENGHIEN pour le climat d'amitié qu'ils n'ont cessé d'entretenir et pour les nombreux encouragements qu'ils n'ont cessé de prodiguer.

1. Introduction.

1.1 Identification du problème à traiter.

Beaucoup de problèmes scientifiques ou autres nécessitent une place mémoire considérable, et par conséquent, créent de très grosses difficultés du point de vue de la gestion de la mémoire. Ainsi, les logiciels qui y sont rattachés se révèlent souvent beaucoup moins performants qu'on ne pouvait l'espérer.

Effectivement, c'est une application mise en oeuvre au département de mathématique de Namur qui nous a conduits à présenter ce mémoire. Quelques doctorants travaillent sur des sujets de mécanique céleste et traitent analytiquement des séries de Poisson, fonctions définies de la manière suivante :

$$F = \quad coef.x_1^{i_1}\ldots x_n^{i_n} {}^{\cos}_{\sin}(j_1u_1+\ldots+j_mu_m)$$

où les x_i et u_i désignent des variables.

Ces utilisateurs disposent pour cela d'un manipulateur de séries (dont le nom est M.S.) appelé à traiter aussi bien un grand nombre de petites séries que quelques séries comportant beaucoup de termes (jusque 15000 termes dans certains cas).

Les opérations accessibles sont la somme et le produit de deux séries ainsi que la multiplication par un scalaire et la dérivation partielle par rapport à l'une ou l'autre des variables. Bien entendu, les possibilités de création, d'effacement, de stockage sur disque des séries sont aussi incluses dans le logiciel ainsi qu'un accès terme à terme.

Bien vite, l'utilisation de M.S. fait apparaître le phénomène suivant : pour des traitements faisant intervenir de très grosses séries, le système ne fait plus que de la pagination et se dégrade rapidement. C'est pourquoi, pour tenter d'améliorer cette situation, nous avons envisagé la création d'un manipulateur d'un bloc de données.

Le problème est donc de créer un logiciel qui gèrerait au mieux l'espace virtuel de l'utilisateur. Ce dernier n'aurait pas à s'en préoccuper et pourrait de la sorte manipuler des blocs de données d'une taille quelconque, éventuellement supérieure à celle de la mémoire virtuelle. Pour la réalisation de ce logiciel, nous essayerons de nous attacher plus particulièrement aux deux critères suivants :

- l'efficacité c'est-à-dire la rapidité d'accès à un bloc

- la transparence vis-à-vis de l'utilisateur.

Nous désirons donc agrandir l'espace virtuel alloué à l'utilisateur en y englobant, en quelque sorte,les fichiers liés à l'application qu'il traite. Nous proposons donc de tenter de régler tous les problèmes inhérents à l'accès à un item d'un bloc de données, en gérant notamment les transferts entre la mémoire virtuelle et les fichiers de l'utilisateur.

1.2 Développement d'un projet-cadre.

1.2.1 Hiérarchie des données.

1. Découper les données en objets logiques accessibles à l'utilisateur

   - l'item représente la plus petite quantité d'information à laquelle l'utilisateur peut accéder en une seule fois (nous ne nous occupons pas de l'accès au contenu de l'item)

   - le bloc représente l'entité de base : c'est un ensemble d'items numérotés ; en se basant sur l'exemple des séries, l'expérience montre que, lorsqu'un terme d'une série est modifié, il y a beaucoup de chance que d'autres termes le

soient également. Ainsi, lors d'une demande d'accès à un item, il faudra s'arranger pour mettre le bloc dans un état qui fasse que les accès suivants à n'importe quel autre item de ce bloc s'exécutent rapidement.

- le fichier représente un ensemble de blocs: les blocs d'un même fichier ont en principe une certaine relation logique entre eux mais celle-ci n'est pas traduite de facon formelle dans une caractéristique informatique. Un exemple illustre cela : un fichier regroupant plusieurs séries de Poisson qui interviennent dans la résolution d'un problème de mécanique céleste. Le dénominateur commun de ces séries, c'est precisément ce problème mathématique et cette relation logique qui les lie n'implique pas nécessairement une relation privilégiée au point de vue des opérations à effectuer. Les blocs sont regroupés dans des fichiers multi-blocs et ce pour ne pas accroître de facon prohibitive le nombre de fichiers manipulés dans le cadre d'une application déterminée.

2. Découper les données en objets physiques c'est-à-dire découper les fichiers et les blocs en pages.

3. Créer un moniteur qui gérerait l'espace de la mémoire virtuelle dont dispose l'utilisateur, cette tâche incluant l'arbitrage des exigences conflictuelles de place mémoire provenant de plusieurs blocs.

   - Exemple illustrant ce cas : l'espace alloué par le système d'exploitation à l'utilisateur est rempli et l'application que ce dernier traite nécessite une nouvelle réservation de place mémoire pour un bloc de données.

1.2.2 Traitements concernés.

Il convient de décrire ici l'architecture des traitements en tenant compte de la hiérarchisation des données, étant donné que cette structuration correspond à l'image que l'utilisateur se fait des données.

- Une première classe de traitements définira donc des fichiers de données :
  il faut donner à l'utilisateur la possibilité de créer des nouveaux fichiers et (ou) de signaler au système l'existence de fichiers qu'il compte utiliser dans le cadre de son application.

- Une deuxième classe concernera des blocs de données :
  d'après les objectifs énoncés ci-dessus, nous pouvons considérer le bloc comme ressource allouable à la demande : l'utilisateur peut vouloir y accéder ou au contraire le libérer ; en outre, il peut lui accorder une certaine priorité ou le copier sur un autre ou encore le vider de son contenu entièrement ou partiellement.

- Une troisième classe de traitements gravitera autour de la notion logique d'item :
  elle regroupe tout ce qui touche à :

  * l'accès d'un item

  * la modification d'un item

- Enfin, préalablement à tous ces traitements, nous prévoyons la création d'un processus de mise en route du manipulateur qui sera chargé de l'initialisation de certains paramètres, de certaines

tables et de la réservation de la place nécessaire pour le stockage des données.

1.2.3 Schéma de la solution proposée.

Un bloc peut se trouver dans deux états distincts :

- un état actif signifiant qu'il a été activé

- un état passif signifiant qu'il a été désactivé ou qu'il n'a pas encore été activé.

Toute utilisation d'un bloc de données s'initialise toujours par une activation de ce bloc. Cette opération permet, par la suite, d'accéder assez vite à ce bloc que ce soit pour une création, une modification ou une consultation.
Une fois que les opérations sur un bloc sont terminées, l'utilisateur est engagé à le désactiver (quitte à le réactiver plus tard) de facon à simplifier la tâche au manipulateur. L'effet d'une désactivation est de libérer la place occupée par le bloc et de supprimer les dispositions permettant un accès rapide et aisé. En d'autres mots, le bloc, tant qu'il n'est pas actif, ne constitue pas une ressource accessible à l'utilisateur.

En outre, nous introduisons la notion de priorité d'un bloc de données. Dès que l'utilisateur a assigné une certaine priorité au bloc, les exigences de ce bloc pour la réservation d'une partie de la mémoire deviennent prioritaires par rapport à des exigences analogues d'autres blocs de priorités inférieures. Toute demande d'accès à une partie de la mémoire est toujours acceptée et si, d'aventure il n'y a plus de place allouable, le bloc émettant la demande jouit d'un droit de préemption sur les pages suffisamment âgées et possédées par les blocs de basses priorités.

Il convient donc, à ce niveau, de distinguer la nature fonctionnelle de la priorité du bloc (liée à l'importance que l'utilisateur lui accorde dans le

contexte du problème qu'il traite) de la nature dynamique de la priorité de page (liée à l'historique de la page au sens où on l'entend dans les systèmes d'exploitation). Ainsi, ce n'est pas parce qu'une page n'a plus été accédée depuis longtemps qu'elle possède une priorité moindre, vu que sa priorité ne dépend plus seulement de son historique, mais aussi de la priorité du bloc la contenant et qui joue d'une certaine facon, le rôle d'un facteur de pondération. Dans un contexte donné, nous espèrons éviter par ce biais le plus possible les transferts de pages entre la mémoire centrale et la mémoire secondaire.

1.3 Première analyse de la solution choisie.

Dans le cadre de ce paragraphe, nous nous proposons de procéder à une découpe relativement sommaire de la solution et d'en définir les lignes principales. Nous considérons les trois parties suivantes :

1.3.1 La gestion des fichiers.

Ceci concerne l'organisation et la gestion de l'espace constitué par les fichiers de l'utilisateur. Chaque fichier contient un certain nombre de blocs de données, ainsi que les dictionnaires donnant des renseignements quant aux endroits du fichier qui sont libres et aux endroits alloués aux blocs. Précisons à ce stade que chaque fichier est quadrillé en pages :

- toutes les pages d'un même bloc sont répertoriées dans un dictionnaire

- toutes les pages d'un même dictionnaire sont chaînées par chaînage simple.

1.3.2 Gestion de la mémoire virtuelle allouée à l'utilisateur.

L'état d'occupation de cet espace virtuel est renseigné dans une carte dont chaque entrée correspond à une page virtuelle et comprend tous les paramètres d'identification de cette page (à condition qu'elle soit reservée, sinon l'entrée est vide), notamment sa priorité et la priorité du bloc qui la contient.

Nous distinguons deux types de pages :

- les pages virtuelles constituant les données effectives d'un bloc : la borne inférieure de leur priorité est la priorité du bloc et la borne supérieure une priorité maximale fixée.

- les pages virtuelles constituant une page d'une table de pages d'un bloc [cette table indique les pages occupées par ce bloc], appelées, par convention, pages "directory" : leur priorité est supérieure d'une unité à la priorité maximale fixée et ce pour les distinguer des pages du type précédent.

Une routine périodique parcourt séquentiellement la carte et diminue d'une unité les priorités des pages occupées, à l'exception des pages "directory" auxquelles elle ne touche pas et des pages de données dont la priorité est égale à celle de leur bloc.

L'allocation s'effectue selon les principes suivants :

- si une page est encore libre, elle est allouée.

- si aucune page n'est libre, est libérée et allouée la page virtuelle qui jouit de la plus petite priorité parmi celles de priorité inférieure à la priorité maximale augmentée de un [si plusieurs pages ont la priorité minimale, est allouée la page dont le bloc qui la contient a la priorité la plus basse.]

- si toutes les pages virtuelles sont occupées par des pages "directory", l'application se termine sur un message d'erreur.

1.3.3 Accès à un bloc et à ses items.

Cet accès s'effectue en deux grandes étapes:

- La première consiste en la mise en place d'un dispositif qui permettra par la suite l'accès effectif aux items du bloc:d'un point de vue logique,elle a pour effet de faire passer un bloc d'un état passif dans un état actif,ce changement rendant directement accessibles et disponibles à l'utilisateur les ressources constituées par les items du bloc.
  En bref,la procédure est la suivante:

  1. A chaque bloc est associé un identificateur qui peut prendre deux formes distinctes:

     * si le bloc se trouve à l'état passif,l'identificateur contient le numéro du fichier qui contient le bloc ainsi que le numéro de ce bloc.

     * si le bloc est dans l'état actif,l'identificateur contient un pointeur vers le début de la table des pages du bloc et le nombre d'items de ce bloc.

  2. le passage à l'état actif consiste donc à:

     * créer en memoire virtuelle la table des pages virtuelles de ce bloc, table qui indiquera les pages occupées par ce bloc (initialement, cette table est vide).

* modifier l'identificateur du bloc.

La seconde étape est caractérisée par l'accès à un(des) item(s) d'un bloc.

Cet item est repéré par un numéro logique qui permet de le localiser dans une page logique du bloc.Nous consultons dès lors la table des pages virtuelles de ce bloc afin de connaître la page virtuelle correspondant à la page logique contenant l'item désiré. Si cette page logique ne se trouve pas encore en mémoire, nous effectuons son transfert et nous notons cette opération dans la table des pages virtuelles du bloc en question.

2. ANALYSE FONCTIONNELLE .

Dans ce chapitre, nous nous proposons de cerner les niveaux distincts dont est composé le problème général décrit au chapitre précédent, ainsi que les différentes fonctions ou primitives intervenant dans la réalisation de la solution.

Dans la mesure du possible,nous tenterons de suivre une démarche descendante par raffinements successifs en vérifiant chaque fois que la primitive créée relève bien d'un des niveaux de découpe déterminés au départ de la présente analyse.

Cette description sera donc abordée suivant cinq étapes successives:

- description des données, des tables et des fichiers. .

- description schématique de la solution.

- détermination des niveaux de découpe.

- détermination des primitives et description "black-box":spécification de leur action (ce qu'elles font et non comment elles le font), de leurs arguments en entrée et en sortie.

- description de l'enchaînement dynamique de ces différentes fonctions.

2.1 DESCRIPTION DES DONNéES, DES TABLES ET DES FICHIERS.

2.1.1 DICTIONNAIRE DES DONNéES.

Ce dictionnaire reprend tous les éléments utilisés dans la suite et toute occurence ultérieure d'un de ces éléments dans le texte fera toujours référence à ce dictionnaire.

1. Informations relatives au bloc.

- blkid:
  identificateur d'un bloc

  * si le bloc est dans un état passif,cet identificateur reprend:

    - le numéro du fichier qui contient le bloc,

    - le numéro de ce bloc,

    - un indicateur notifiant que le bloc se trouve dans un état passif.

  * sinon,cet identificateur reprend:

    - un pointeur vers le début de la table des pages virtuelles de ce bloc,

    - le nombre d'items de ce bloc,

    - un indicateur notifiant que le bloc se trouve dans un état actif.

- blid

  adresse d'un bloc actif:c'est le pointeur vers le début de la table des pages virtuelles de ce bloc.

- niblk

  nombre d'items du bloc.

- prblk

  priorité du bloc.

- blk

  numéro du bloc.

- flag

  indice mis à un ou à zéro suivant l'état du bloc et contenu dans l'identificateur du bloc.

2. Informations relatives au fichier.

- tf

  ce numéro de fichier désigne une entrée de la table des fichiers.

- finame

  nom de fichier.

- nbfibl

  nombre de blocs d'un fichier.

- nbfipg

nombre de pages d'un fichier.

- fp

  numéro d'une page d'un fichier.

- ffp

  numéro de la page "fichier" chaînée avec la page "fichier" no fp.

- nbfiut

  nombre de fichiers définis par l'utilisateur.

3. Informations relatives aux pages.

- dirpg

  numéro logique d'une page "directory" c'est-à-dire une page d'une table des pages d'un bloc.

- datpg

  numéro logique d'une page de données d'un bloc.

- pgdim

  dimension d'une page (en mots).

- nbdipg

  nombre de pages "directory" d'un bloc c'est-à-dire le nombre de pages de la table des pages d'un bloc.

- prpg

  priorité d'une page virtuelle.

- primax

  priorité maximale d'une page virtuelle.

- le

  dernière entrée active d'une page "directory" (c'est-à-dire la dernière entrée qui contient encore des renseignements concernant les pages d'un bloc).

4. Informations relatives à l'item.

- item

  numéro logique d'un item dans un bloc.

- itdim

  dimension d'un item (en mots).

5. Informations relatives aux tables.

a. Table des fichiers.

- tfi

  désignation de la table des fichiers.

- adtf

  adresse en mémoire virtuelle du début de la table des fichiers.

- finame

  nom du fichier renseigné dans une entrée active de la table tfi.

- sem

  nombre de blocs actifs du fichier.

- jfn

  numéro d'identification interne du fichier.

b. Table des pages virtuelles d'un bloc.

- tpb1

  désignation de la table des pages virtuelles d'un bloc.

- adr1

  pointeur vers une page de la table des pages virtuelles d'un bloc.

- pt1

  pointeur vers une entrée de la table des pages virtuelles d'un bloc.

- fadr1

  pointeur vers la page de la table des pages virtuelles d'un bloc qui suit la page virtuelle d'adresse adr1 de la même table des pages virtuelles.

c. Table des pages sur disque d'un bloc.

- tpb2

  désignation de la table des pages sur disque d'un bloc.

- adr2

  pointeur vers une page virtuelle contenant une page de

la tables des pages sur disque d'un bloc.

- pt2

  pointeur vers une entrée en mémoire virtuelle de la table des pages sur disque d'un bloc.

- fadr2

  pointeur vers la page virtuelle de la table des pages sur disque d'un bloc qui suit la page sur disque (de la même table des pages) implantée en mémoire virtuelle à l'adresse adr2.

d. Table des pages libres d'un fichier.

- tplf

  désignation de la table des pages libres d'un fichier.

- adtplf

  pointeur vers la fenêtre destinée à recevoir la dernière page de la table tplf d'un fichier.

- lent

  nombre de pages libres qui sont référencées dans une page d'une table tplf.

- pt

  pointeur de chaînage inverse entre les pages d'une table tplf.

e. Carte ou table des pages de la mémoire virtuelle.

- adca

  pointeur vers le début de la carte.

- nbpgmv

  nombre de pages virtuelles demandées par l'utilisateur pour y stocker ses données.

Les autres informations relatives à cette carte ont déjà été décrites ci-dessus.

2.1.2 DESCRIPTION DES FICHIERS.

Comme nous l'avons déjà souligné, les fichiers sont considérés comme étant "multi-blocs" et les pages d'une même table des pages sont rangées suivant la technique d'un chaînage simple. Les pages des fichiers sont numérotées. L' entrée no i de la table des pages d'un bloc contient un pointeur vers la page "fichier" contenant la page logique no i de ce bloc. Cette procédure fait que l'organisation et la gestion des pages du fichier s'avère relativement souple.

L'entête des fichiers est constituée de renseignements généraux suivis de la dernière page de la table des pages libres et de chacune des premières pages des tables des pages des blocs appartenant à ces fichiers.
Un fichier est donc structuré comme suit:
Pour simplifier le schéma, supposons que le fichier contient deux blocs :

(1) : première page du fichier : dernière page de la table des pages libres du fichier;

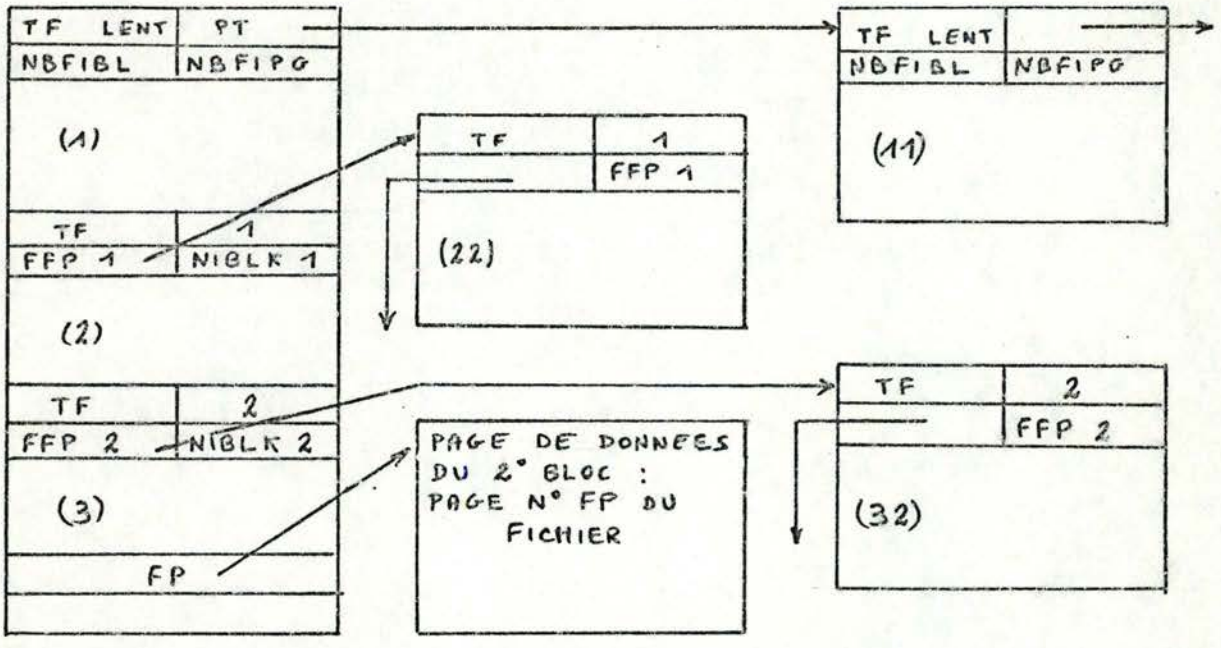
(11) : page n° PT du fichier : avant dernière page de la table des pages libres du fichier;

(2) : deuxième page du fichier : première page de la table des pages du premier bloc;

(22) : page n° FFP1 du fichier : deuxième page de la table des pages du premier bloc;

(3) : troisième page du fichier : première page de la table des pages du deuxième bloc;

(32) : page n° FFP2 du fichier : deuxième page de la table des pages du deuxième bloc.



Nous remarquons donc qu'il est constitué, comme cela avait été précisé précédemment:

- des données.

- de la table des pages libres du fichier.

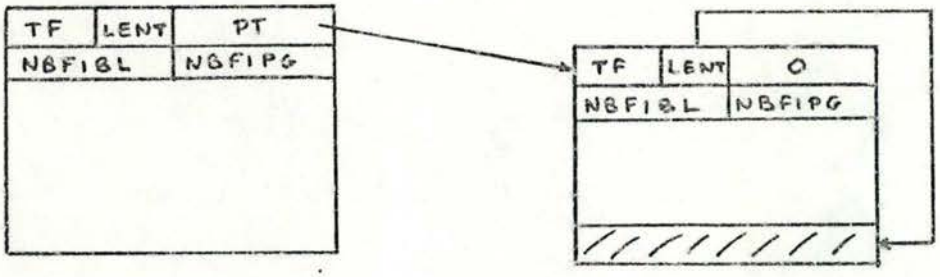
- des tables des pages "fichier" des blocs.

2.1.3 DESCRIPTION DES TABLES.

1. Table des pages libres d'un fichier.

Cette table peut être considérée comme une pile et d'ailleurs elle est gérée comme telle. Les pages "fichier" de cette table sont reliées entre elles par chaînage inverse, la dernière page pointant vers l'avant-dernière et ainsi de suite. Par convention, dans la

première page, ce pointeur prend la valeur nulle. La dernière page de cette table est implantée à une place fixe dans le fichier (la première page) et peut être décrite comme suit:
Supposons que cette table comporte deux pages :



Si la table des pages libres comporte plus d'une page, il est clair que toutes les pages de cette table,sauf la dernière, sont remplies.

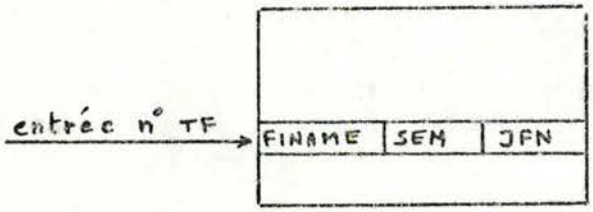
Quant à la représentation de cette table en mémoire virtuelle, nous procédons comme suit:

une page virtuelle d'adresse adtplf est réservée à l'usage de la table tplf; nous pouvons la considérer comme une fenêtre sur cette table des pages libres qui contient toujours la dernière page de la table tplf d'un des fichiers de l'utilisateur.

2. Table des fichiers.

Cette table est résidente en mémoire virtuelle et contient les renseignements principaux concernant les fichiers que l'application de l'utilisateur fait intervenir.

Nous pouvons raisonnablement admettre qu'elle peut tenir dans une seule page virtuelle, mais ce n'est pas du tout une restriction.



Cette table est remplie à l'initialisation du système à l'aide des renseignements fournis par l'utilisateur.

Pour la signification des symboles, cfr supra 2.1.1.5.

3. Carte ou table des pages de la mémoire virtuelle.

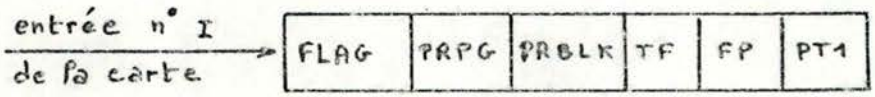
Cette table, comme la précédente, résidera toujours en mémoire virtuelle, son début étant spécifié par l'adresse adca.Elle contient tous les renseignements quant à la configuration et l'occupation de l'espace virtuel.

Elle comporte autant d'entrées qu'il y a eu de pages virtuelles réservées par l'utilisateur à l'usage de ses blocs de données , ceci étant consigné dans la variable nbpgmv.

Trois cas dictincts peuvent se présenter:

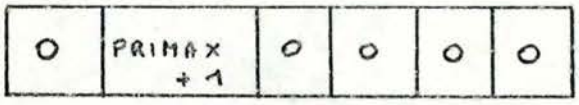
- la page virtuelle est libre, auquel cas l'entrée correspondante de la carte est mise à zéro.

- la page virtuelle est occupée par une page de données d'un bloc: l'entrée correspondante de la carte a le numéro i et prend la forme:



ce qui signifie que la page virtuelle no i jouit d'une priorité prpg, appartient à un bloc de priorité prblk, est référencée dans une table des pages virtuelles d'un bloc à l'adresse ptl et correspond, en fait, à la page no fp du fichier référencé dans l'entrée no tf de la table des fichiers.Enfin, l'indicateur flag est mis à zéro ou à un suivant que la page n'a pas été modifiée ou l'a déjà été.

- la page virtuelle est occupée par une page "directory" d'un bloc: l'entrée correspondante de la carte est décrite comme suit:



ce qui signifie que la page virtuelle jouit de la priorité maximale incrémentée de un.

4. Table des pages d'un bloc.

Dans les schémas qui suivent, nous utilisons la convention suivante:

qui signifie que le symbole x désigne l'adresse du mot pointé par la flèche.

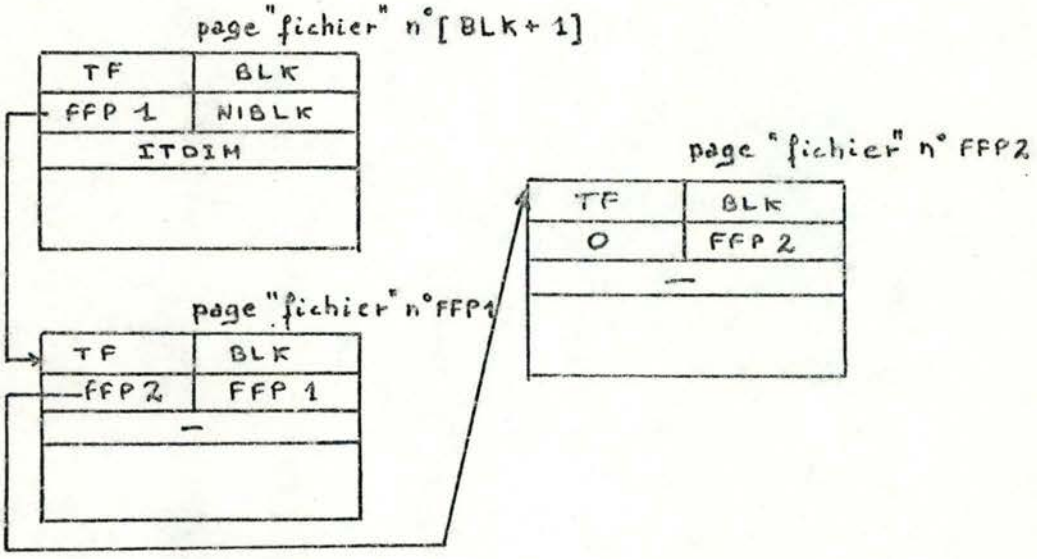
- configuration de cette table sur fichier.

La table tpb2 se trouve toujours dans le fichier qui contient le bloc qu'elle référence. La première page de le table tpb2

est implantée dans la page "fichier" dont le numéro est celui du bloc référencé, augmenté de un.

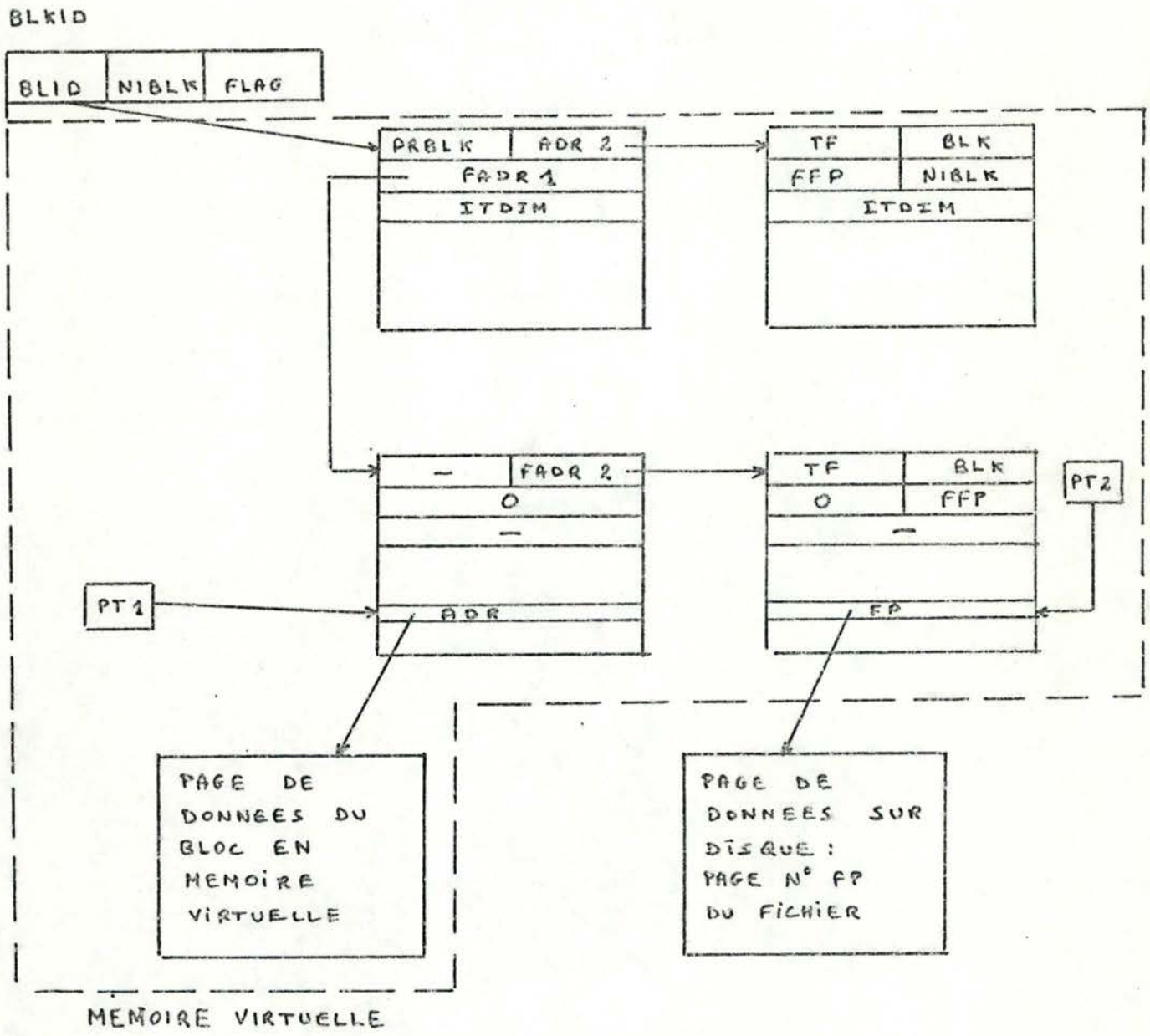
Sa structure est la suivante:

Supposons que cette table contient trois pages :



- configuration de cette table en mémoire virtuelle.

  Pour faciliter la compréhension du schéma, supposons que la table des pages du bloc comporte deux pages.



Remarquons que la partie droite du schéma ci-dessus est une copie en mémoire virtuelle de la table telle qu'elle existe dans le fichier et que la partie gauche contient toutes les informations permettant un accès rapide et aisé aux items du bloc concerné. La priorité du bloc est uniquement reprise dans l'entête de la première page de la table tph1, de même que le nombre d'items du bloc est seulement mentionné dans la première page de la table tph2.

Pour la dernière page de la table tph1, le pointeur de

chaînage vers la page suivante [fadrl] est, par convention, mis à zéro. Les pages de la table tpbl sont chaînées entre elles et le chaînage entre les pages virtuelles de la table tpb2 s'effectue indirectement entre pages correspondantes des tables tpbl et tpb2.

2.2 DESCRIPTION SCHéMATIQUE DE LA SOLUTION.

2.2.1 GESTION DES FICHIERS.

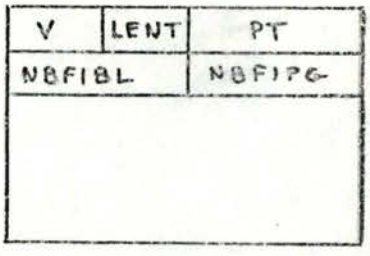
Cette gestion est assurée via la table des pages libres.Nous allons examiner les deux situations pouvant se présenter:

1. Allocation d'une page "fichier" connaissant le numéro du fichier [tf].

   [*]

   Nous savons qu'une fenêtre est réservée pour recevoir la dernière page de la table des pages libres, encore faut-il vérifier qu'elle contient la dernière page de la table tplf du fichier sur lequel nous travaillons.

   Rappelons sa structure:



   Le contenu de la variable v est le numéro d'une entrée de la table tf:

s'il est différent de tf, cela signifie que nous avons affaire à la dernière page d'une table tplf d'un autre fichier;il faut par conséquent la recopier sur la première page du fichier en question (cfr supra 2.1.2.) et transférer dans la fenêtre la dernière page de la table tplf du fichier qui nous intéresse.

Deux cas peuvent se présenter:

- la page n'est pas vide, c'est-à-dire que lent est distinct de 0.

  Elle référence donc encore des pages libres du fichier; la page allouée est indiquée par le pointeur lent qu'on décrémente d'une unité.

- la page est vide, c'est-à-dire que lent est nul.

  Elle ne référence plus aucune page libre du fichier:

  * si il y a plus d'une page dans la table tplf [pt distinct de 0] , alors la page pointée par pt est lue dans la fenêtre et la page "fichier" pointée par pt est allouée.

  * si il y a une seule page dans la table tplf [pt nul] , cela signifie que le fichier est rempli;une nouvelle page lui est adjointe et le nombre de ses pages est incrémenté de un.

2. Désallocation d'une page "fichier" dont on connaît le numéro [fp].

Nous supposons que la dernière page de la table tplf se trouve en mémoire virtuelle(sinon la procédure [*] expliquée ci-dessus est encore d'application).

A nouveau,nous pouvons rencontrer les deux cas suivants:

- la fenêtre est remplie:elle est recopiée sur la page "fichier"

qui devait être désallouée et une nouvelle page de la table des pages libres du fichier est initialisée en mettant à zéro les pointeurs lent et pt.

- la fenêtre n'est pas remplie: le pointeur lent est incrémenté d'une unité et la valeur fp est affectée à l'entrée no lent de la fenêtre en question.

2.2.2 GESTION DE LA MéMOIRE VIRTUELLE.

La configuration de la mémoire virtuelle (c'est-à-dire la facon dont elle est occupée) est illustrée par la carte (cfr supra 2.1.3.3.). Rappelons brièvement la structure d'un élément de la carte:

- si la page correspondante est vide, l'élément est nul.

- si la page correspondante est une page "directory", l'élément est de la forme:

0	PRIMAX+1	0	0	0	0

- sinon, les renseignements suivants sont disponibles:

FLAG	PBLK	PPG	TF	FP	PT1

1. Allocation d'une page virtuelle.

La carte est parcourue séquentiellement et la page allouée est celle dont la priorité est minimale.

Plusieurs cas peuvent se présenter:

- le pointeur ptl est nul, ce qui signifie que la page allouée est libre: le problème est donc réglé.

- le pointeur ptl est non nul,ce qui signifie que l'espace virtuel est entièrement occupé et que, en l'occurence, la page allouée est déjà occupée par une page de données d'un bloc. L'indicateur [flag] est consulté afin de savoir si cette page de données a subi une modification, auquel cas il faut la recopier sur fichier (à l'aide des éléments tf et fp).Ensuite, l'entrée (d'adresse ptl) de la table tpbl du bloc concerné est mise à zéro ainsi que l'élément de la carte. De la sorte, la page virtuelle initialement choisie se révèle enfin libre et elle peut être allouée.

2. Désallocation d'une page virtuelle.

L'entrée correspondante de la carte est simplement affectée d'une valeur nulle.

2.2.3 ACCèS à UN BLOC ET à SES ITEMS.

1. Première étape.

Il s'agit de mettre le bloc dans un état actif c'est-à-dire:

a. amener en mémoire virtuelle le début de la table tpb2 du bloc.

b. créer le début de la table tpbl du bloc.

c. modifier l'identificateur du bloc.

d. assigner une priorité au bloc et donner une dimension à ses items.

Précisons la nature de ces différentes actions.

L'identificateur du bloc est de la forme:

- Nous nous proposons de charger en mémoire virtuelle la première page de la table tpb2 du bloc. A l'aide de la variable tf, nous pouvons accéder, via la table des fichiers, au nom du fichier contenant le bloc en question:
  l'entrée no tf de la table des fichiers contenant notamment le nom du fichier.
  Le numéro du bloc dans ce fichier nous est fourni par la variable blk. Se référant à l'organisation des fichiers (cfr supra 2.1.2.) ,nous pouvons dire que la première page de la table tpb2 du bloc est contenue dans la page "fichier" dont le numéro d'identification n'est rien d'autre que le numéro du bloc [blk] augmenté de un.Il reste donc à transférer cette page en mémoire virtuelle.

- La seconde action à réaliser est la création de la première page de la table tpb1 du bloc.
  Ce processus s'exécute de la facon suivante:

  * réservation d'une page virtuelle [*],

  * mise à jour de la carte,

  * création de l'entête de la page:

    - chaînage avec la première page de la table tpb2 du bloc,

- mise à zéro du pointeur de chaînage avec la page suivante de la table tpb1 (vu que pour le moment cette table ne comporte qu'une page).

- Pour remplacer l'identificateur et indiquer que le bloc est dans l'état actif, nous devons disposer de deux informations:

  * l'adresse du début de la table tpb1 obtenue lorsqu'une page virtuelle est réservée pour la première page de la table tpb1 (cfr supra [*])

  * le nombre d'items du bloc: ceci se trouvant dans la première page de la table tpb2.

- La priorité du bloc ainsi que la dimension de ses items sont consignées respectivement dans la table tpb1 et dans les tables tpb1 et tpb2.Une précision à propos de la dimension de l'item : le contenu du troisième mot de la table tpb2 est d'abord testé:

  * si il est nul,cela signifie que le bloc vient d'être créé et la dimension des items de ce bloc est alors sauvée dans le mot en question.

  * si il est non nul, il est comparé avec la dimension de l'item qui a été communiquée au système et si il n'y a pas égalité, le traitement est arrêté et un message d'erreur est envoyé: l'utilisateur a donné une dimension de l'item différente de celle déclarée à la création du bloc.

2. Deuxième étape.

Il s'agit d'accéder à un item d'un bloc.

Les arguments en entrée sont:

- l'identificateur du bloc actif: blkid ,c'est-à-dire:

  * un pointeur vers le début de la table tpbl: blid

  * le nombre d'items du bloc: nibLk

- le numéro logique de l'item: item

Cette étape s'exécute suivant quatre phases successives.

- Calcul du numéro de la page logique du bloc contenant l'item et localisation de l'item dans la page en question.
  Ces opérations sont illustrées par les relations:
  datpg = (item - 1)/(pgdim/itdim) + 1
  et de = (item - ((datpg - 1)*nbitpg) - 1)*itdim.
  La dimension des items du bloc [itdim] est consignée dans la première page de la table tpb2 et la dimension d'une page [pgdim] est définie à l'initialisation du système.

- Calcul du numéro de la page logique "directory" référencant la page logique de données contenant l'item cherché.
  ce calcul est effectué grâce à la relation:
  dirpg = (datpg - 1)/(pgdim - 3) + 1.

- Recherche de la page virtuelle "directory" correspondant à la page logique "directory" de la phase précédente.
  Deux cas peuvent alors se présenter:

  * la page logique "directory" ne se trouve pas encore dans la chaîne de la table tpbl:

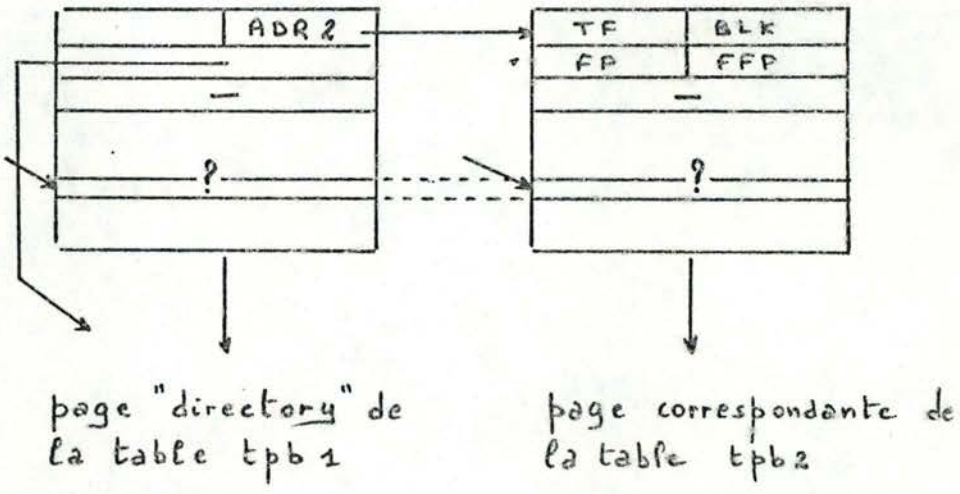
il faut donc compléter cette chaîne,

- si la version "fichier" de la page logique "directory" existe,cela est réalisé grâce à la table tpb2

- sinon (cas où une nouvelle page logique de données est créée et que la dernière page "directory" de la table tpb2 est remplie), une nouvelle page de la table tpb2 et la page correspondante de la table tpb1 sont créées.

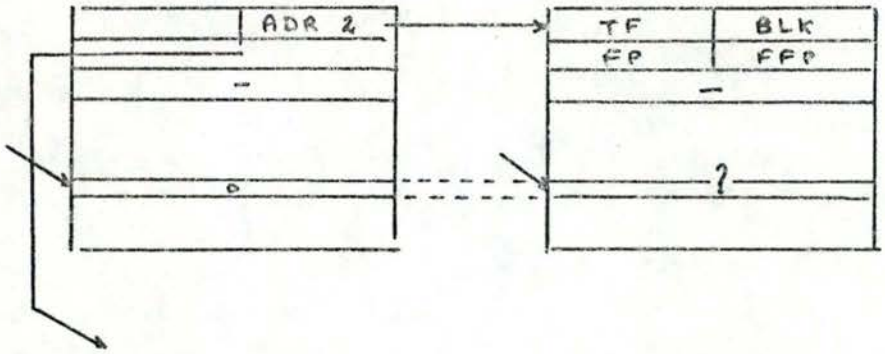
* la page logique "directory" est déjà implantée dans la chaîne de la table tpb1 et il n'y a alors aucun problème.

- Recherche de la page virtuelle contenant la page logique de données (contenant l'item désiré).
L'entrée de la page virtuelle "directory" (cfr phase précédente) correspondant à la page logique de données est examinée c'est-à-dire schématiquement:



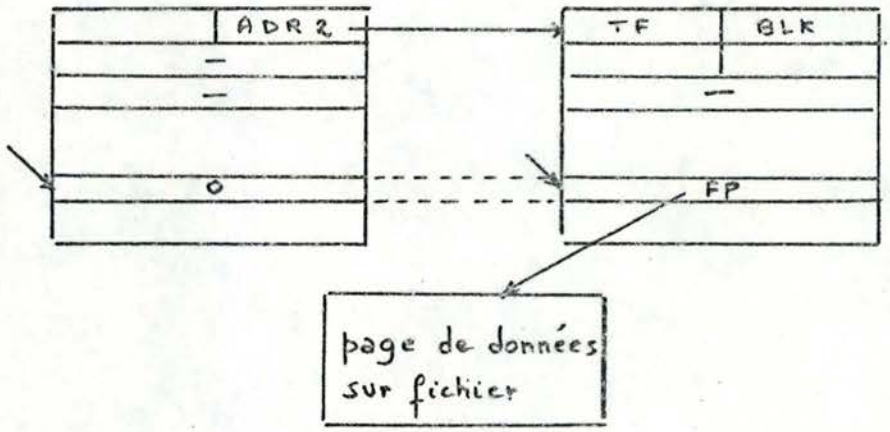
* si l'entrée est vide, l'entrée correspondante dans la

chaîne de la table tpb2 est examinée:



- si l'entrée est à nouveau vide, alors une page "fichier" et une page virtuelle sont allouées et leurs numéros respectifs sont notés dans les entrées respectives des tables tpbl et tpb2 et nous nous sommes ainsi ramenés au cas [*] (cfr ci-dessous).

- sinon, nous nous trouvons dans une situation schématisée comme suit:

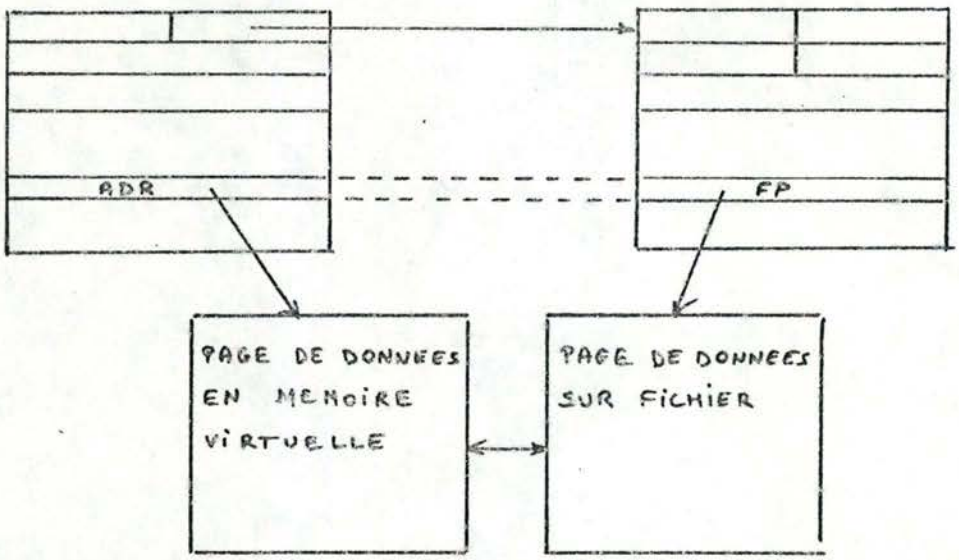


la page "fichier" no fp est transférée dans une page virtuelle dont l'adresse est notée dans la table tpbl et nous retrouvons à nouveau le cas [*] (cfr ci-dessous).

* [*]

sinon,nous obtenons le schéma suivant qui se passe de

tout commentaire:



2.3 DéTERMINATION DES NIVEAUX DE LA DéCOUPE FONCTIONNELLE.

Nous avons tenté de suivre une démarche de type "top-down", afin d'obtenir une découpe logique et modulaire. Avant de pousser plus loin cette analyse, nous explicitons les principes qui nous ont conduits à adopter une telle découpe.Nous considérons qu'un bloc est constitué de deux types de pages:

- les pages de données qui contiennent les items logiques du bloc,

- les pages "directory" qui contiennent les pages des tables des pages du bloc (c'est-à-dire les tables tpb1 et tpb2).

Ainsi,la "directory" d'un bloc n'est donc rien d'autre que la réunion des deux tables des pages tpb1 et tpb2.Tout cela nous a amenés à distinguer les niveaux de découpe suivants:

- le niveau 0 est constitué de tout ce qui est mis à la disposition de l'utilisateur.

- le niveau 1 est caractérisé par le fait que l'entité logique manipulée est soit la table (les tables tp1f, tpb2,...), soit la

"directory" d'un bloc, soit le bloc de données.

- le niveau 2 est le niveau où l'objet de référence est la page logique de données ou la pagé logique de "directory".

- le niveau 3 est le niveau où apparaît l'insertion des données en mémoire virtuelle ou sur fichier. La page logique y est considérée sous la forme d'une page insérée en mémoire virtuelle ou d'une page résidente sur fichier.

- le niveau 4 est le niveau physique dont relèvent tous les utilitaires de de la page.

2.4 DéCOMPOSITION EN FONCTIONS ET DESCRIPTION.

Partons des traitements cités dans le paragraphe 1.2. et qui relèvent bien évidemment du niveau 0.Tâchons de les expliciter formellement et de les spécifier clairement et correctement.

2.4.1 TRAITEMENTS CONCERNANT L'ITEM.

1. Lecture d'un item d'un bloc.

   Désignation:ITEMRD(blkid,item,var)

   input:

   - blkid : identificateur d'un bloc.

   - item : numéro d'un item.

output:

- var : vecteur quelconque.

Fonctions:

- vérifier si le numéro de l'item existe bien (sinon envoyer un message d'erreur et arrêter le traitement)

- lire l'item (dont le numéro logique est spécifié par item) du bloc identifié par blkid et le mettre dans le vecteur var.

2. Ecriture ou modification d'un item d'un bloc.

Désignation:ITEMWR(blkid,item,var)

input:

- blkid :identificateur d'un bloc.

- item :numéro d'un item.

- var :vecteur quelconque.

output:aucun.

Fonctions:

- vérifier si le numéro de l'item est correct c'est-à-dire:

  * dans le cas de la modification d'un item, vérifier si le numéro existe bien .

  * dans le cas de la création d'un item, vérifier si le numéro est bien égal au nombre d'items du bloc concerné augmenté de un.

- modifier ou créer l'item (dont le numéro est item) du bloc identifié par blkid avec le contenu du vecteur var.

2.4.2 TRAITEMENTS CONCERNANT LE BLOC.

1. Activation d'un bloc.

Désignation: BLKACT(blkid,prblk,itdim)

input:

- blkid :identificateur d'un bloc.

- prblk :priorité du bloc.

- itdim :taille des items du bloc.

output:aucun.

Fonctions:

- vérifier si le bloc est bien dans un état passif (sinon envoyer un message d'erreur et arrêter le traitement)

- activer le bloc

- assigner au bloc une priorité donnée par prblk

- assigner aux items du bloc une dimension donnée par itdim si le bloc n'existe pas encore, sinon vérifier la compatibilité entre la dimension communiquée et la dimension existante.

2. Désactivation d'un bloc.

Désignation:BLKDEA(blkid)

input:

- blkid :identificateur de bloc.

output:aucun.

Fonctions:

- vérifier que le bloc est bien dans un état actif (sinon envoyer un message d'erreur et arrêter le traitement)

- désactiver le bloc.

3. Copie du contenu d'un bloc dans un autre.

Désignation:BLKCOP(blkid1,blkid2)

input:

- blkid1:identificateur de bloc.

- blkid2:idem

output:aucun.

Fonctions:

- vérifier si le bloc est actif

- si le bloc identifié par blkid2 est plus grand que l'autre, ramener sa taille à celle de ce dernier

- copier le contenu du premier bloc dans le second.

4. Destruction totale ou partielle du contenu d'un bloc.

Désignation:BLKCUT(blkid,item)

input:

- blkid :identificateur de bloc.

- item :numéro d'un item.

output:aucun.

Fonction:

- vérifier si le bloc est actif

- détruire les items [du bloc identifié par blkid] qui suivent l'item identifié par le numéro logique item.

2.4.3 TRAITEMENTS CONCERNANT LES FICHIERS.

1. Définition et identification d'un fichier.

Désignation:FILDEF(finame,nbfibl,blklst)

input:

- finame:nom du fichier

- nbfibl:nombre de blocs du fichier

- blklst:liste des identificateurs du bloc.

output:aucun.

Fonction:

- créer une entrée dans la table des fichiers et vérifier que le paramètre nbfibl décrit ci-dessus représente bien le nombre

exact de blocs du fichier en question.

- initialiser les identificateurs des blocs du fichier.

2. Création d'un fichier.

Désignation:FILCRE(finame,nbfibl,blklst)

input:

- finame:nom du.fichier

- nbfibl:nombre de blocs du fichier

- blklst:liste des identificateurs du bloc.

output:aucun.

Fonction:

- créer un fichier multi-blocs dont le nom et le nombre de blocs seront respectivement le contenu des paramètres finame et nbfibl (étant supposé que ce fichier n'existe pas encore).

- initialiser les identificateurs des blocs du fichier.

2.4.4 INITIALISATION DU SYSTèME.

Désignation:SYSINI(nbpgmv,nbfiut)

input:

- nbpgmv:dimension (exprimée en pages) de l'espace virtuel demandé par l'utilisateur pour le stockage de ses blocs de données

- nbfiut:nombre maximal de fichiers utilisés

output:aucun.

Fonctions:

- réserver la place mémoire nécessaire à l'implantation de la carte , de la table des fichiers et du buffer pour la table des pages libres des fichiers.

- réserver la place mémoire nécessaire pour le stockage des blocs de données.

2.4.5 ANALYSE ET DéCOUPE DE CES TRAITEMENTS.

Nous allons maintenant essayer d'exprimer ces différents traitements à l'aide de primitives relevant des niveaux de découpe définis précédemment. Pour cela, passons en revue ces traitements et examinons-les d'un peu plus près.

1. L'analyse de la fonction BLKACT nous incite à créer une fonction de niveau 1 qui a pour but d'activer la "directory" du bloc que l'on veut faire passer à l'état actif,c'est-à-dire en la spécifiant:

   Désignation:DIRACT(blkid)

   input:

   - blkid :identificateur de bloc.

   output:aucun.

   Fonction:

   - activer la "directory" du bloc.

   L'activation de la "directory" consiste donc à:

   - amener en mémoire virtuelle la première page de la table tpb2 du bloc identifié par blkid

- créer en mémoire virtuelle la première page de la table tpbl du bloc identifié par blkid

- modifier l'identificateur

Cette activation peut donc être réalisée grâce aux trois primitives de niveau 4:

- allocation d'une page virtuelle [PGMGET].

- mise à jour de la carte [PGMSMP].

- lecture d'une page "fichier" sur une page virtuelle [PGFIRD].

Les spécifications respectives de ces fonctions sont:

- Désignation:PGMGET(adr)

  input:aucun.

  output:

    * adr :adresse d'une page virtuelle.

  Fonction:

    * allouer une page virtuelle et mettre son adresse dans la variable adr.

- Désignation:PGMSMP(adr,flag,prpg,prblk,tf,fp,ptl)

  input:

    * adr :adresse d'une entrée de la carte

    * flag :variable indiquant si la page référencée dans cette entrée de la carte a été ou non modifiée

    * prpg :priorité de la page référencée

* prblk :priorité du bloc contenant cette page

* tf :numéro du fichier qui la contient

* fp :numéro de la page "fichier" qui lui correspond

* ptl :pointeur vers l'entrée de la table tpbl où la page est référencée.

fonction:

* mettre à jour l'entrée de la carte d'adresse adr avec les valeurs des arguments en entrée.

- Désignation:PGFIRD(tf,fp,adr)

input:

* tf :numéro d'un fichier

* fp :numéro d'une page de ce fichier

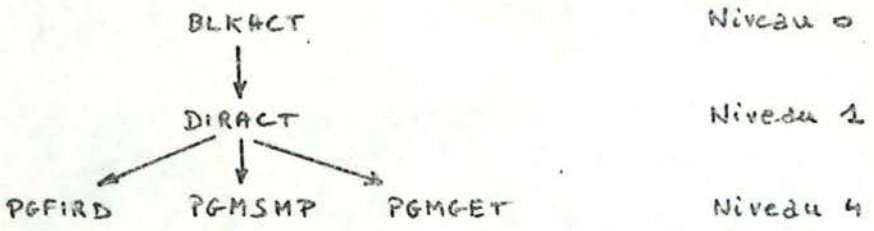
* adr :adresse d'une page virtuelle.

output:aucun.

Fonction:

* amener dans la page virtuelle d'adresse adr la page no fp du fichier no tf.

La fonction BLKACT peut donc être décrite par le schéma suivant:



3. La méthode d'accès à un item d'un bloc actif [en lecture ou en écriture] a été décrite au paragraphe 2.2.3. Il en ressort clairement que la difficulté principale réside dans la recherche de la page virtuelle contenant une page logique donnée. Cette fonction est bien de niveau 2 [traitement de pages logiques] et est désignée par: DATADR. Cette tâche en utilise une autre de niveau 3 [désignée par DATIN] qui est chargée d'amener en mémoire virtuelle la page de données [à moins qu'elle ne s'y trouve déjà] et de noter son adresse dans la table tpbl du bloc concerné. La fonction DATADR nécessite (cfr supra 2.2.3.) l'apport d'une fonction parallèle de niveau 2 [identifiée par DIRADR] chargée de rechercher la page virtuelle contenant une page logique "directory", cette fonction DIRADR faisant elle-même appel à un traitement de niveau 3 [désigné par DIRIN] ayant pour but d'amener en mémoire virtuelle une page "directory" [à moins qu'elle n'y soit déjà]. Les fonctions de niveau 3 DATIN et DIRIN étant sensées transférer des pages "fichier" en mémoire virtuelle (et éventuellement créer des pages "fichier") utilisent les fonctions de niveau 4 décrites précédemment (cfr supra 2.4.5.1) :PGMGET, PGMSMP et PGFIRD et une nouvelle fonction s'insérant dans le même niveau: allocation d'une page "fichier" [PGFGET].

Nous pouvons dès lors passer à une description complète des différentes fonctions que nous avons relevées:

- Désignation: DATADR(blid,datpg,adr)

  * blid :pointeur vers la table tpbl d'un bloc.

  * datpg :numéro d'une page logique de données de ce bloc.

  output:

  * adr :pointeur vers une page virtuelle.

  Fonction:

  * fournir l'adresse de la page virtuelle contenant la page logique [no datpg] du bloc pointé par blid.

- Désignation:DATIN(blid,pt1,pt2)

  input:

  * blid :pointeur vers le début de la table tpbl d'un bloc.

  * pt1 :pointeur vers une entrée de la table tpbl du bloc.

  * pt2 :pointeur vers l'entrée correspondante de la table tpb2.

  output:aucun.

  Fonctions:

  * examiner l'entrée [pointée par pt1] de la table tpbl du bloc identifié par blid.

  * si elle contient une adresse, le traitement est fini.

  * sinon,examiner l'entrée [pointée par pt2]de la table tpb2 du bloc identifié par blid

    - si cette entrée est vide, allouer une page

"fichier",noter son numéro dans l'entrée de la table tpb2 et ensuite,allouer une page virtuelle et noter son numéro dans l'entrée de la table tpbl

- sinon,amener en mémoire virtuelle la page de données du bloc identifié par blid, cette page étant référencée dans l'entrée [pointée par pt2] de la table tpb2 de ce bloc et noter son adresse virtuelle dans l'entrée de la table tpbl pointée par ptl.

- Désignation:DIRADR(blid,dirpg,adr)

input:

* blid :pointeur vers le début de la table tpbl.

* dirpg :numéro d'une page logique "directory".

output:

* adr :adresse d'une page virtuelle.

Fonction:

* fournir l'adresse de la page virtuelle contenant la page logique "directory" [no dirpg] de la table tpbl du bloc pointé par blid.

- Désignation:DIRIN(adrl,adr2)

input:

* adrl :pointeur vers la dernière page d'une table tpbl.

* adr2 :pointeur vers la dernième page virtuelle de la table tpb2 correspondante.

output:aucun.

Fonctions:

* amener en mémoire virtuelle la page "fichier" "directory" qui suit la page "fichier" "directory" implantée en mémoire virtuelle à l'adresse adr2 (ou, à défaut, si celle-ci est la dernière de la table tpb2, créer une nouvelle page "fichier" "directory" et créer la page virtuelle "directory" correspondante)

* créer la page "directory" correspondante de la table tpb1

* mettre dans adr1 et adr2 les adresses des deux pages ainsi créées.

- Désignation: PGFGET (tf,adr)

input:

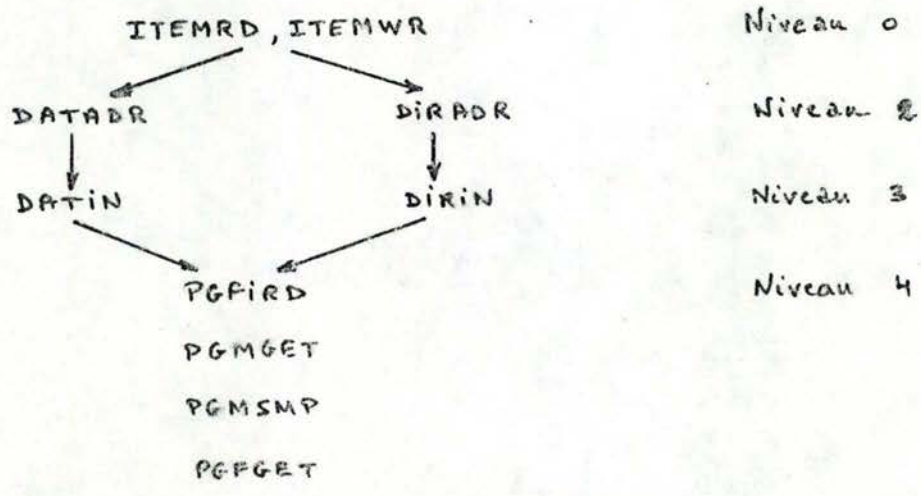
* tf : numéro d'identification logique d'un fichier.

output:

* fp : numéro de la page "fichier" qui a été allouée.

Fonction: allouer une page du fichier no tf.

La constitution interne des fonctions ITEMRD et ITEMWR se schématise donc par:



3.2. La désactivation d'un bloc comporte deux sous-fonctions principales de niveau 1:

- la désactivation des données

- la désactivation de la "directory".

La fonction DATDEA est chargée de désactiver les données. Elle relève du niveau 1 et peut être considérée comme la désactivation en cascade des pages de données implantées en mémoire virtuelle. Ainsi, la désactivation d'une page de données sera effectuée par une fonction de niveau 3 appelée DATOUT. Il est clair que ce traitement DATOUT utilisera les deux fonctions suivantes:

- libérer une page virtuelle, tâche assumée par la fonction de niveau 4 identifiée par PGMFRE

- transférer sur fichier une page virtuelle, tâche assumée par la fonction de niveau 4 identifiée par PGFIWR.

Un raisonnement tout à fait analogue peut être suivi pour la désactivation de la "directory" du bloc et ce en parlant cette fois de la désactivation des pages "directory" . Les fonctions correspondantes relevant des niveaux 1 et 3 sont respectivement identifiées par les noms DIRDEA et DIROUT. Nous pouvons donc passer aux spécifications de ces différentes fonctions.

- Désignation:DATDEA(blkid)

  input:

    * blkid :identificateur d'un bloc.

  output:aucun.

  Fonctions:

    * recopier sur fichier les pages virtuelles occupées par

les données du bloc identifié par blkid et ensuite, les désallouer.

- Désignation:DATOUT(blid,pt1,pt2)

input:

* blid :pointeur vers le début d'une table tpb1

* pt1 :pointeur vers une entrée de la table tpb1 du bloc pointé par blid

* pt2 :pointeur vers l'entrée correspondante de la table tpb2 du même bloc.

output:aucun.

Fonctions:

* transférer sur fichier la page de données du bloc identifié par blid et référencée dans l'entrée [d'adresse pt1] de la table tpb1 du bloc

* désallouer la page virtuelle occupée par cette page de données

- Désignation:PGMFRE(adr)

input:

* adr :adresse d'une page virtuelle

output:aucun.

Fonction:

* désallouer la page virtuelle d'adresse adr.

- Désignation:PGFIWR(tf,fp,adr)

input:

* tf :numéro d'un fichier

* fp :numéro d'une page de ce fichier

* adr :adresse d'une page virtuelle

output:aucun.

Fonction:

* transférer sur la page no fp du fichier no tf le contenu de la page virtuelle d'adresse adr.

- Désignation:DIRDEA(blkid)

input:

* blkid :identificateur d'un bloc.

output:aucun.

Fonctions:

* transférer sur fichier les pages virtuelles occupées par les pages de la table tpb2 du bloc identifié par blkid

* désallouer les pages virtuelles occupées par les pages des tables tpb1 et tpb2 du même bloc

* modifier l'identificateur pour indiquer que le bloc est dans l'état passif.

- Désignation:DIROUT(adr1,adr2)

input:

* adr1 :adresse virtuelle d'une page "directory" d'une

table tpb1

* adr2 :adresse virtuelle d'une page "directory" d'une table tpb2

output:

* arguments identiques indiquant les adresses des pages virtuelles "directory" suivantes

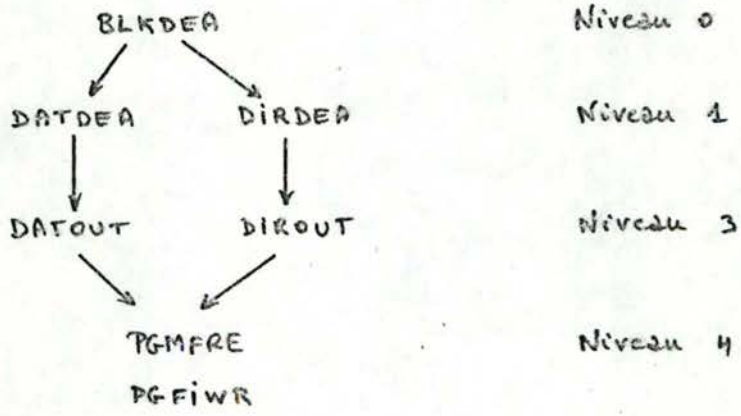
Fonctions:

* copier sur fichier la page d'adresse adr2

* désallouer les pages d'adresses adr1 et adr2

* sauver dans adr1 et adr2 les adresses des pages "directory" suivantes dans les tables tpb1 et tpb2

L'analyse de la fonction BLKDEA est ainsi terminée. Résumons par un petit schéma:



3. Quant à la copie du contenu d'un bloc dans un autre, on peut dire que les primitives suivantes seront utilisées:

- Calculer le nombre de pages logiques "directory" d'une table tpbl d'un bloc.

Ce calcul est effectué par la fonction DIRDIM qui relève du niveau 2, puisqu'elle manipule des pages logiques "directory". La logique interne de cette fonction est élémentaire et il n'est pas nécessaire de s'y attacher plus longuement.

- Tronquer le bloc à partir d'une certaine page de données.

Nous définissons pour cela la fonction de niveau 2 identifiée par DATCUT. Cette fonction consistant à détruire des pages de données ,il faut donc connaître les pages "directory" où elles sont référencées afin de pouvoir les localiser.Nous pouvons donc dégager les primitives suivantes:

* calculer l'adresse virtuelle d'une page "directory", ceci étant réalisé par la fonction DIRADR déjà explicitée,

* détruire une page de données référencée dans une entrée d'une page "directory", cette action étant prise en charge par une fonction de niveau 3 appelée DATDS.

- Tronquer la "directory" d'un bloc à partir d'une certaine page logique "directory".

Cette primitive s'insère dans le niveau 2 et est réalisée par la fonction DIRCUT. Cette fonction manipule les concepts

* de page "directory" dont il faut connaître l'adresse: cela s'avère possible grâce à la primitive DIRADR déjà commentée

* de page virtuelle "directory" et page "fichier" "directory" qu'il faut détruire:cette tâche est assumée par un traitement de niveau 3 appelé DIRDS .

Il est clair que ce traitement s'exécute en faisant appel à des primitives du type: désallocation d'une page "fichier" [PGFFRE] et désallocation d'une page virtuelle [PGMFRE] ,cette dernière ayant déjà été explicitée. Nous faisons remarquer que la découpe effectuée pour la troncature de la "directory" s'avère en tous points identique à celle qui a été proposée plus haut pour la troncature des données.

- Copier les pages de données d'un bloc, référencées dans une page "directory" de ce bloc, dans les pages de données d'un autre bloc et mettre à jour la "directory" de ce bloc.

Cela peut être réalisé par une fonction de niveau 2; appelons-là DATCOP. Cette fonction traite des pages logiques "directory" dont elle doit calculer l'adresse ainsi que des pages logiques de données dont elle doit disposer en mémoire virtuelle.Ces deux actions sont réalisées par des primitives que nous avons définies antérieurement et qui sont respectivement désignées par DIRADR et DATIN.

Formalisons maintenant toutes les fonctions qui ont été dégagées à l'issue de cette analyse de la fonction BLKCOP.

- Désignation:DIRDIM(blid,nbdipg)

input:

* blid :pointeur vers le début de la table tphl d'un bloc.

output:

* nbdipg:nombre de pages de la "directory" du bloc.

Fonction:

* fournir la taille de la "directory" du bloc identifié par blid.

- Désignation:DATCUT(blid,datpg)

input:

* blid :pointeur vers le début de la table tpbl d'un bloc

* datpg :numéro d'une page logique de données de ce bloc

output:aucun.

Fonction:

* détruire les pages logiques de données du bloc pointé par blid et ce à partir de la page logique no (datpg + 1).

- Désignation:DATDS(blid,ptl,pt2)

input:

* blid :pointeur vers le début de la table tpbl d'un bloc

* ptl :pointeur vers une entrée de la table tpbl de ce bloc

* pt2 :pointeur vers l'entrée correspondante de la table tpb2 de ce bloc

output:aucun.

Fonctions:

* désallouer la page "fichier" occupée par la page logique de données du bloc pointé par blid et référencée dans l'entrée d'adresse pt2 de la table tpb2,

* désallouer la page virtuelle éventuellement occupée par cette page logique de données et référencée, dans ce cas, dans l'entrée d'adresse ptl de la table tpbl.

- Désignation:DIRCUT(blid,dirpg)

input:

* blid :pointeur vers le début de la table tpbl d'un bloc.

* dirpg :numéro d'une page logique "directory".

output:aucun.

Fonction:

* détruire les pages logiques "directory" du bloc pointé par blid et ce à partir de la page no (dirpg + 1).

- Désignation:DIRDS(adr1,adr2)

input:

* adr1 :adresse d'une page virtuelle "directory" d'une table tpbl

* adr2 :adresse d'une page virtuelle "directory" d'une table tpb2

output:arguments identiques qui désignent les adresses des pages virtuelles "directory" suivantes.

Fonctions:

* désallouer la page virtuelle "directory" [d'adresse adr1] de la table tpbl,

* désallouer la page virtuelle "directory" [d'adresse pt2] de la table tpb2,

* désallouer la page "fichier" "directory" correspondant à la page "directory" [d'adresse adr2] de la table tpb2,

* fournir dans adr1 et adr2 les adresses des pages virtuelles "directory" suivantes dans les tables tpb1 et tpb2.

- Désignation:DATCOP(blid1,blid2,dirpg,nbdapg)

input:

* blid1 :pointeur vers le début de la table tpb1 d'un bloc

* blid2 :pointeur vers le début de la table tpb1 d'un deuxième bloc

* dirpg :numéro d'une page logique "directory"

* nbdapg:nombre de pages de données référencées dans la page "directory" dont le numéro logique est donné par dirpg.

output:aucun.

Fonction:

* copier les pages logiques de données du bloc pointé par blid1 et référencées dans la page logique no dirpg de la "directory" de ce bloc [il y a nbdapg pages de données qui sont référencées dans cette page "directory"] dans les pages logiques correspondantes [de même numéro] du bloc pointé par blid2,

* mettre à jour la page "directory" no dirpg du bloc

identifié par blid2.

- Désignation:PGFFRE (tf,fp)

input:

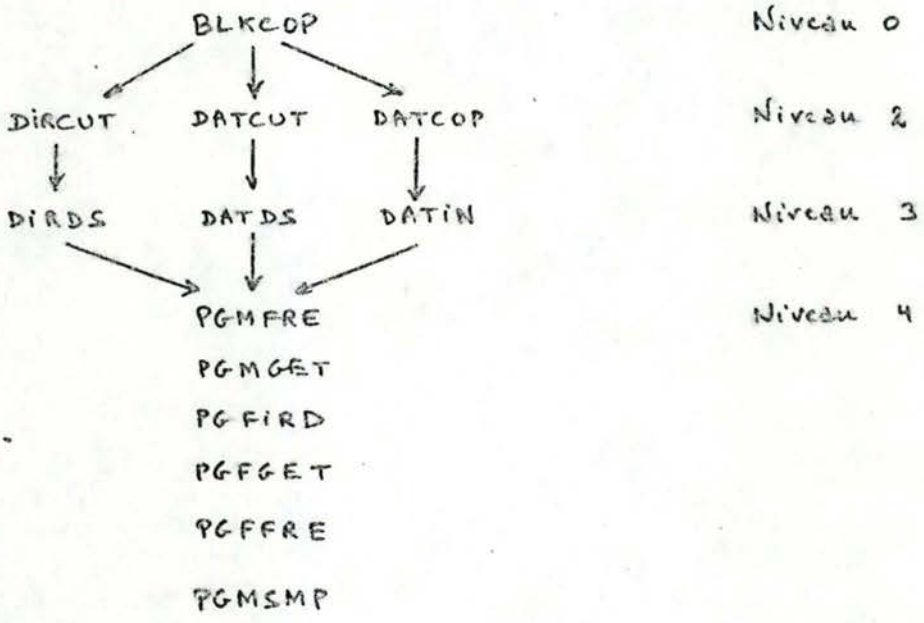
* tf : numéro d'identification logique d'un fichier.

* fp :numéro de la page de ce fichier qui va être désallouée.

output:aucun.

Fonction: désallouer la page no fp du fichier no tf.

La logique interne de la fonction BLKCOP est illustrée par le schéma:



4. La troncature d'un bloc se compose des deux actions principales:

- troncature des données,

- troncature de la "directory"

primitives assumées par les fonctions DATCUT et DIRCUT que nous avons déjà étudiées.

5. La déclaration d'un fichier engendre la séquence suivante d'actions:

   - recherche d'une entrée libre dans la table des fichiers et assignation d'un numéro d'identification interne à ce fichier: cette action peut s'insérer dans le niveau 1 et nous l'appelerons FILINI.

   - activation du fichier c'est-à-dire l'ouverture du fichier: cela est réalisé par la fonction de niveau 1 appelée FILACT.

   - lecture du nombre de blocs dont est constitué le fichier et comparaison avec la taille [exprimée en blocs] du fichier telle que l'utilisateur l'a déclarée, cette primitive étant assurée par la fonction de niveau 1 identifiée par OFISET.

   - désactivation du fichier c'est-à-dire fermeture du fichier, cela étant pris en charge par la fonction de niveau 1 FILDEA.

   - initialisation des identificateurs, primitive élémentaire qui ne fait l'objet d'aucun commentaire.

Nous pouvons dès lors écrire les spécifications des fonctions rencontrées ci-dessus.

- Désignation:FILINI (finame,tf,flag)

  input:

  * finame : nom du fichier

  * flag :indicateur mis à 0 ou à 1 selon que le fichier existe déjà ou va être créé.

  output:

* tf : numéro d'identification logique du fichier.

Fonctions:

* rechercher une entrée libre de la table des fichiers.

* donner un numero d'identification interne au fichier.

* consigner cela dans l'entrée définie ci-dessus.

- Désignation:FILACT(tf)

input:

* tf : numéro d'identification logique du fichier.

output:aucun.

Fonction:ouvrir le fichier no tf.

- Désignation:OFISET (tf,nbfibl)

input:

* tf : numéro d'identification logique d'un fichier.

* nbfibl : taille [exprimée en blocs] du fichier.

output:aucun.

Fonctions:

* lecture dans le fichier du nombre de ses blocs et comparaison avec le paramètre nbfibl :si il n'y a pas égalité, envoi d'un message d'erreur.

- Désignation:FILDEA (tf)

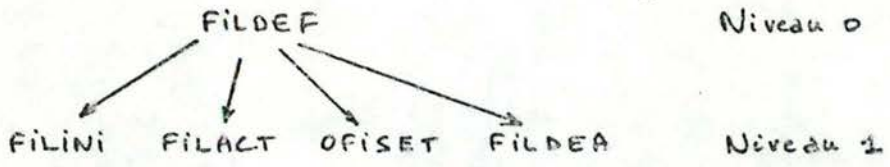
input:

* tf : numéro d'identification logique d'un fichier.

output:aucun.

Fonction: fermer le fichier no tf.

Le schéma suivant résume le comportement de la fonction FILDEF:



6. De même, la création d'un fichier ne s'avère guère plus difficile: tout au plus s'agit-il de créer dans le fichier la table des pages libres et les entêtes des tables des pages des blocs le formant. Ainsi, au sein de cette fonction, les actions à exécuter sont analogues excepté pour la fonction OFISET à laquelle se substitue la fonction NFISET spécifiée comme suit:

Désignation:NFISET (tf,nbfibl)

input:

- tf : numéro d'identification logique d'un fichier.

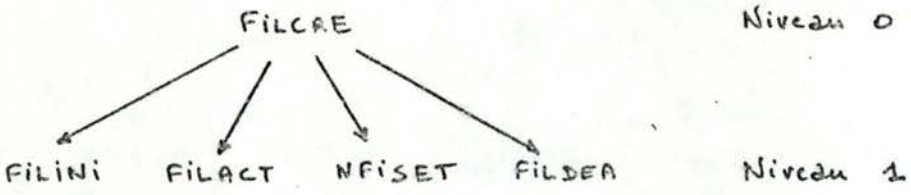
- nbfibl : nombre de blocs de ce fichier.

Fonctions:

- créer la première page de la table des pages libres du fichier no tf.

- créer la première page de la table tpb2 de chacun des blocs de ce fichier.

La fonction FILCRE peut être représentée schématiquement de la facon suivante:



Enfin, l'initialisation du système se compose des fonctions suivantes:

- réserver la place mémoire nécessaire aux données de l'utilisateur:

  Désignation:ALLOCA(nbpgmv,adr)

  input:

  * nbpgmv:nombre de pages virtuelles

  output:

  * adr : adresse du début de la zone qui a été réservée.

  Fonction:

  * réserver un nombre de pages virtuelles égal à nbpgmv et sauver l'adresse de début de cette zone.

- créer la carte:

  Désignation:MAPCRE(nbpgmv,adr)

  input:

  * nbpgmv:nombre de pages virtuelles

  output:

  * adr : adresse virtuelle du début de la carte.

  Fonction:

* réserver la place mémoire nécessaire à l'implantation de la carte , cette place étant proportionnelle au nombre de pages virtuelles exigées par l'utilisateur [nbpgmv]

* sauver l'adresse de cette table.

- créer la table des fichiers:

Désignation:TFICRE(nbfiut,adr)

input:

* nbfiut:nombre de fichiers déclarés par l'utilisateur.

output.

* adr : adresse de début de la table des fichiers.

Fonction:

* réserver la place mémoire nécessaire à l'implantation de la table des fichiers

* sauver l'adresse de cette table

- créer la fenêtre réservée à la dernière page de la table des pages libres du fichier courant:

Désignation:TPLFCR(adr)

input:aucun.

output.

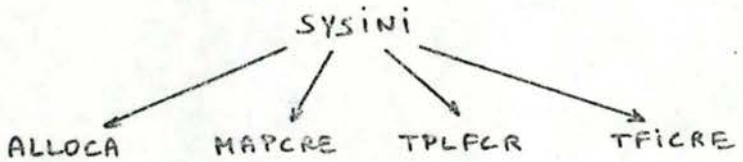
* adr : adresse de début de la table tplf.

Fonction:

* réserver une page virtuelle

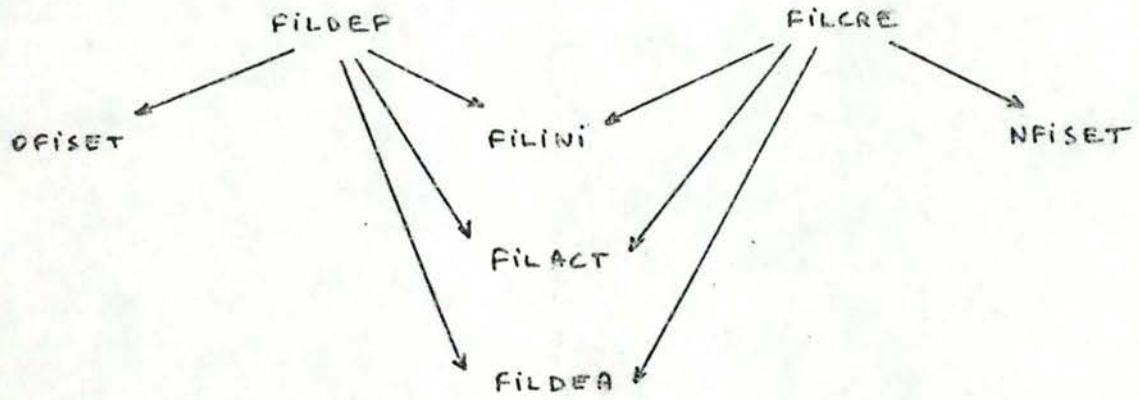
* sauver son adresse

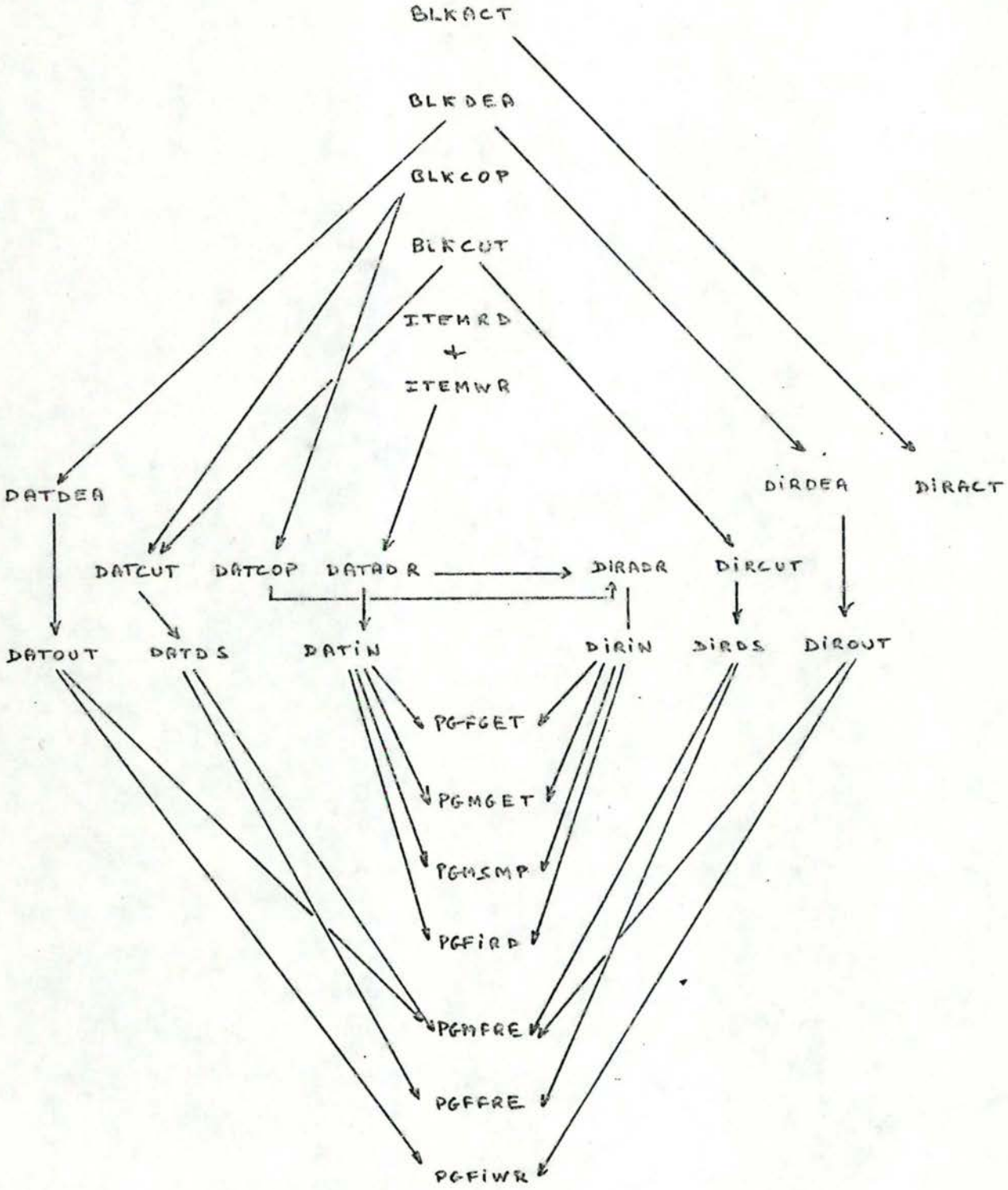
La fonction initialisant le système se schématise donc comme suit:



Cela étant dit, nous avons donc terminé le tour d'horizon des outils qui sont mis à la disposition de l'utilisateur.

En guise de conclusion à ce chapitre, nous nous proposons de résumer l'analyse de la découpe en fonctions dans le diagramme suivant, qui décrit l'enchaînement des primitives ainsi définies:




BLKACT
BLKDEA
BLKCOP
BLKCUT
ITEMRD
ITEMWR
DATDEA
DIRDEA
DIRACT
DATCUT
DATCOP
DATADR
DIRADR
DIRCUT
DATOUT
DATDS
DATIN
DIRIN
DIRDS
DIROUT
PGFGET
PGMGET
PGMSMP
PGFIRD
PGMFRE
PGFFRE
PGFIWR

3. Dossier de programmation.

Ce chapitre est consacré avant tout à l'implémentation du manipulateur décrit de facon fonctionnelle au chapitre précédent.
Il comprend d'une part les programmes proprements dits avec, à l'appui, commentaires et spécifications et d'autre part un manuel destiné à l'utilisateur et un dictionnaire reprenant toutes les notations utilisées.
Pour des raisons de portabilité, nous avons choisi de programmer l'essentiel dans un langage de haut niveau et pour des raisons d'efficacité , quelques routines spécifiques ont été rédigées en MACRO-ASSEMBLEUR. Le choix du langage de haut niveau s'est porté sur FORTRAN (c'est-à-dire la version FORTRAN-20 de chez Digital) de facon à ce que les routines du manipulateur puissent être appelées par des programmes rédigés en FORTRAN, beaucoup d'utilisateurs potentiels ne connaissant que ce langage.

3.1 DESCRIPTION DE LA LIBRAIRIE DES PROGRAMMES PROPOSES A L'UTILISATEUR.

Cette librairie est susceptible d'intéresser des utilisateurs travaillant sur des applications traitant des masses importantes de données. Nous agencons ces données de la facon suivante: l'unité logique est l'item, les items sont regroupés en blocs et les blocs en fichiers. La dimension des items d'un bloc est identique et ne peut varier en cours d'utilisation , à moins qu'un autre bloc ne soit recopié sur ce bloc auquel cas la taille des items devient celle des items du bloc "source". La structure interne de l'item n'intervient pas dans la conception du manipulateur et elle est définie par l'utilisateur en fonction de l'application qu'il traite: l'item pouvant être une matrice, un vecteur,...
Neuf routines sont mises à la disposition de l'utilisateur. En voici une description succinte à laquelle sont jointes les modalités d'utilisation et les inévitables restrictions qui sont imposées.

1. SUBROUTINE SYSINI (nbpgmv,nbfiut,primax)

   a. Arguments:

- Nbpgmv:variable entière désignant le nombre de pages de données dont l'utilisateur aura besoin.

- Nbfiut:variable entière donnant le nombre maximal de fichiers qui seront accédés ou créés durant l'exécution du programme de l'utilisateur.

- Primax:variable entière strictement positive désignant la priorité maximale qui peut être accordée à une page de données.

b. Fonctions:

réserver le nombre de pages nécessaires au stockage des données de l'utilisateur et effectuer toutes les opérations qui permettront par la suite aux autres routines décrites un peu plus loin d'exécuter correctement la tâche qui leur incombe.

c. Messages d'erreurs: la réservation de la zone pour recevoir les données n'a pas pu se faire ou la réservation de place pour les tables internes au manipulateur s'est avérée impossible.

d. Commentaire:lors de toute utilisation du manipulateur de blocs, la routine SYSINI doit absolument être appelée en premier lieu et aucune autre routine ne pourra se dérouler normalement si la phase d'initialisation n'a pas été réalisée.

2. SUBROUTINE FILCRE (finame,nbfibl,blklst)

a. Arguments:

- finame est un vecteur entier à trois dimensions contenant le nom d'un fichier qui doit comprendre au plus 14 caractères.

- nbfibl est une variable entière indiquant le nombre de blocs du fichier.

- blklst est un vecteur entier qui reprend la liste des identificateurs des blocs du fichier.

b. Fonction: créer un fichier identifié par le nom contenu dans le vecteur finame et destiné à contenir nbfibl blocs identifiés par les noms de la liste blklst.

c. Messages d'erreurs.

- l'initialisation du système n'a pas été effectuée.

- l'ouverture [ou la fermeture] du fichier n'a pu s'effectuer correctement.

- la transformation du nom du fichier en code asciz a échoué.

- le fichier existe déjà, alors qu'il a été déclaré comme n'existant pas encore.

- la recherche du jfn du fichier a échoué.

- une erreur s'est produite lors d'une opération d'entrée/sortie.

d. Commentaires:cette routine ne peut être utilisée que pour des nouveaux fichiers . Il faut prendre garde au fait que si le nombre de blocs déclarés [nbfibl] est différent de la dimension de la liste des identificateurs [blklst], l'erreur n'est pas détectée à ce niveau et le programme continue à se dérouler comme si de rien n'était.

3. SUBROUTINE FILDEF (finame,nbfibl,blklst)

a. Arguments:identiques à ceux de la routine FILCRE.

b. Fonction: déclarer que le fichier identifié par le nom contenu dans le vecteur finame existe déjà et sera utilisé dans l'application que l'utilisateur traite ;signaler que ce fichier contient [nbfibl] blocs identifiés par les identificateurs de la liste blklst.

c. Messages d'erreurs:

- l'initialisation du système n'a pas été effectuée.

- l'ouverture [ou la fermeture] du fichier n'a pas pu s'effectuer correctement.

- la transformation du nom du fichier en code asciz a échoué.

- la recherche du jfn du fichier a échoué.

- une erreur s'est produite lors d'une opération d'entrée/sortie.

- le nombre de blocs du fichier n'est pas celui communiqué.

d. Commentaires:cette routine ne peut être utilisée que pour des fichiers existant déjà;en outre, le seconde remarque à formuler est analogue à celle du commentaire concernant la routine FILCRE. Il convient d'insister sur le fait que tout fichier manipulé doit avoir été créé par la routine FILCRE car son organisation interne est étroitement liée au bon fonctionnement du manipulateur.

4. SUBROUTINE BLKACT (blkid,prblk,itdim)

a. Arguments:

- **blkid:** variable entière qui représente l'identificateur d'un bloc.

- **prblk:** variable entière qui désigne la priorité du bloc.

- **itdim:** variable entière qui donne la dimension des items d'un bloc.

b. Fonction:

- mettre le bloc identifié par blkid dans un état actif (l'utilisateur peut ainsi y accéder par la suite),

- lui assigner une priorité prblk,

- définir une dimension itdim pour ses items.

c. Messages d'erreur.

- l'initialisation du système n'a pas été effectuée.

- le bloc est déjà dans un état actif.

- le bloc existe déjà (il a été créé auparavant) et ses items ont une dimension différente de la dimension communiquée [itdim].

- la priorité d'un bloc [prblk] ne peut dépasser à aucun moment la priorité d'aucune de ses pages et par conséquent, elle ne peut être supérieure à la priorité maximale d'une page, ce paramètre étant défini lors de l'initialisation.

d. Commentaires:

L'utilisateur ne peut lire ou modifier ou créer un (ou des) items d'un bloc tant que ce bloc n'est pas dans un état actif: avant d'accéder à un bloc , il doit obligatoirement faire appel à la routine BLKACT.

Si, dans le cadre de l'application qu'il traite, l'utilisateur se rend compte qu'un certain bloc sera plus souvent référencé que les autres , il peut lui assigner une priorité élevée ce qui accroîtra probablement la rapidité d'accès à ce bloc.

5. SUBROUTINE BLKCOP (blkid1,blkid2)

a. Arguments: blkid1 et blkid2 sont des variables entières identifiant deux blocs de données.

b. Fonction: copier le contenu du bloc identifié par blkid1 dans le bloc identifié par blkid2.

c. Messages d'erreurs:

- les deux blocs ne se trouvent pas dans un état actif.

- la phase d'initialisation n'a pas été réalisée.

d. Commentaire: pour pouvoir faire appel à cette routine les deux blocs doivent être activés et ce par la routine BLKACT.

6. SUBROUTINE BLKCUT (blkid,item)

a. Arguments

- blkid est une variable entière identifiant un bloc.

- item est une variable entière spécifiant le numéro logique d'un item.

b. Fonction: tronquer le bloc identifié par blkid à partir des items suivant l' item dont le numéro logique est donné par [item].

c. Messages d'erreurs:

- la phase d'initialisation n'a pas été réalisée.

- le bloc n'est pas dans un état actif.

- le numéro de l'item n'existe pas.

7. SUBROUTINE ITEMRD (blkid,item,var)

a. Arguments

- blkid est une variable entière désignant l'identificateur d'un bloc.

- item est une variable entière donnant le numéro d'un item de ce bloc.

- var est un vecteur quelconque dont la dimension est celle des items du bloc.

b. Fonction:lire l'item (dont le numéro logique est donné par [item]) du bloc identifié par blkid et écrire son contenu dans le vecteur var.

c. Messages d'erreurs: le bloc n'est pas dans un état actif ou le numéro de l'item n'existe pas.

8. SUBROUTINE ITEMWR (blkid,item,var)

a. Arguments

- blkid est une variable entière désignant l'identificateur d'un bloc.

- item est une variable entière désignant le numéro d'un item de ce bloc.

- var est un vecteur quelconque dont la dimension est

celle des items du bloc.

b. Fonction: écrire le contenu du vecteur var dans l'item (dont le numéro logique est donné par item) du bloc identifié par blkid.

c. Messages d'erreurs: le bloc n'est pas dans un état actif ou le numéro de l'item est incorrect.

d. Commentaire: si l'écriture correspond à la modification d'un item existant, le numéro doit être compris entre un et le nombre d'items du bloc et si elle correspond à la création d'un nouvel item, le numéro doit être celui qui suit immédiatement le numéro du dernier item.

9. SUBROUTINE BLKDEA (blkid)

a. Argument: blkid est une variable entière désignant l'identificateur d'un bloc.

b. Fonction: mettre le bloc identifié par blkid dans un état passif, c'est-à-dire sauver sur fichier toutes les modifications effectuées sur ce bloc depuis sa dernière activation.

c. Message d'erreur: le bloc en question est déjà désactivé.

d. Commentaire: la désactivation d'un bloc signifie que l'utilisateur n'en a plus besoin, du moins dans l'immédiat; ainsi, toute référence à un bloc, que ce soit en lecture ou en écriture, doit s'insérer entre un appel à la routine

BLKACT et un appel à la routine BLKDEA. La désactivation est donc à un bloc ce que la fermeture est au fichier.

En résumé, le bloc peut être considéré comme une ressource: elle est accessible à l'utilisateur lorsque celui-ci en a fait la demande à l'aide de la routine BLKACT et elle est relâchée par le biais de la routine BLKDEA. En outre, pour que le bloc soit identifié, il faut que le fichier le contenant ait été déclaré par l'entremise d'une des deux routines FILDE ou FILCRE. Durant la période séparant l'activation et la désactivation d'un bloc, l'utilisateur peut appeler à son gré les routines BLKCUT, BLKCOP, ITEMWR et ITEMRD.

Deux remarques importantes s'imposent:

- toute utilisation du manipulateur commencera toujours par un appel à la routine SYSINI: dans le cas contraire, le programme principal se terminera immédiatement par un message signalant l'erreur.

- pour que les données de l'utilisateur demeurent dans un état cohérent à l'issue d'opérations employant le manipulateur, tous les blocs ayant été activés durant l'exécution d'un traitement doivent absolument faire l'objet d'une désactivation avant la fin de ce traitement sinon les fichiers risquent d'être perdus.

En bref, la séquence normale et correcte des appels sera toujours du type:

```
CALL  SYSINI
CALL  FILCRE
      FILDE
CALL  BLKACT
      CALL  BLKCUT
            BLKCOP
            ITEMRD
            ITEMWR
CALL  BLKDEA
```

3.2 DICTIONNAIRE DES DONNEES.

Nous donnons dans ce paragraphe la liste (dont nous espérons qu'elle sera exhaustive) des variables utilisées dans les programmes constituant le manipulateur de blocs de données.
Pour plus de facilité, nous les avons classées par ordre alphabétique. Voici le dictionnaire de ces variables:

- adl : adresse d'une page "directory" de la table tphl.

- ad2 : idem.

- ad12 : adresse d'une page virtuelle "directory" de la table tph?.

- ad22 : idem.

- ada : adresse d'une page virtuelle.

- adb : idem.

- adc : adresse d'une entrée de la carte.

- adcl : idem.

- adc2 : idem.

- adca : adresse du début de la carte.

- admv : adresse du début de la zone (en mémoire virtuelle) qui est réservée aux données de l'utilisateur.

- adr : adresse d'une page virtuelle.

- adrl : adresse d'une page "directory" de la table tpbl d'un bloc.

- adr2 : adresse d'une page virtuelle "directory" de la table tpb2 d'un bloc.

- adrbuf : adresse de la page virtuelle réservée au buffer des entrées/sorties.

- adtf : adresse du début de la table des fichiers.

- adtplf : adresse de la page virtuelle faisant office de buffer pour la dernière page de la table des pages libres du fichier courant.

- blid : pointeur vers le début de la table tpbl d'un bloc actif.

- blidl : idem.

- blid2 : idem.

- blk : numéro logique d'un bloc dans un fichier.

- blkid : identificateur d'un bloc.

- blkidl : idem.

- blkid2 : idem.

- blklst : liste des identificateurs des blocs d'un même fichier.

- datpg : numéro logique d'une page de données d'un bloc.

- de : numéro logique d'une entrée d'une page "directory" déterminée (localisation d'une page de données) ou d'une entrée d'une page de données (localisation d'un item).

- dim : dimension d'un item ou d'une zone de la mémoire virtuelle.

- dirpg : numéro logique d'une page "directory" d'un bloc.

- ercode : code de retour en cas d'erreur dans l'exécution d'un programme.

- fadr1 : adresse d'une page "directory" de la table tph1 d'un bloc.

- fadr2 : adresse d'une page virtuelle "directory" de la table tph2 d'un bloc.

- ffp : numéro logique d'une page d'un fichier.

- finame : nom d'un fichier.

- flag : variable d'état d'une page de données (la page en question a-t-elle subi des modifications ?) ou d'un fichier (le fichier existe-t-il déjà ou bien va-t-il être créé ?).

- fp : numéro logique d'une page d'un fichier.

- indic : variable renseignant le fait que la phase d'initialisation du système a été ou non réalisée.

- int : variable intermédiaire.

- itdim : dimension des items d'un bloc.

- itdim1 : idem.

- itdim2 : idem.

- item : numéro logique d'un item d'un bloc.

- jfn : numéro d'identification interne d'un fichier ("job file number")

- lent : nombre de pages libres qui sont référencées dans la dernière page de la table des pages libres d'un fichier.

- nb : numéro d'une erreur au sein d'un type déterminé d'erreurs.

- nbdap1 : nombre de pages de données d'un bloc.

- nbdap2 : idem.

- nbdapg : idem.

- nbdipg : nombre de pages "directory" d'un bloc.

- nbfib1 : nombre de blocs d'un fichier.

- nbfipg : nombre de pages d'un fichier.

- nbfiut : nombre maximal de fichiers utilisés dans une application.

- nbitpg : nombre d'items dans une page de données.

- nbpgmv : nombre de pages virtuelles réservées pour les données d'un utilisateur.

- ndipg1 : idem.

- ndipg2 : idem.

- niblk : nombre d'items d'un bloc.

- niblk1 : idem.

- niblk2 : idem.

- np : numéro logique d'une page virtuelle.

- numbuf : numéro logique de la page virtuelle jouant le rôle du buffer des entrées/sorties.

- numfil : nombre maximal de fichiers utilisés dans le cadre d'une application.

- numpg : nombre de pages réservées pour le stockage des données de l'utilisateur.

- pdidim : dimension d'une page "directory".

- pgdim : dimension d'une page de données.

- prblk : priorité accordée à un bloc.

- primax : priorité maximale qui peut être assignée à une page de données.

- prpg : priorité assignée à une page.

- pt : pointeur de chaînage inverse entre les pages d'une table des pages libres d'un fichier.

- pt1 : pointeur vers une entrée d'une page "directory" de la table tpb1 d'un bloc.

- pt2 : pointeur vers une entrée d'une page virtuelle "directory" de la table tpb2 d'un bloc.

- sem : nombre de blocs actifs d'un fichier.

- tf : numéro logique d'une entrée dans la table des fichiers.

Nous pouvons dès lors passer à la description des programmes proprement dits: c'est l'objet du paragraphe suivant.

3.3 PROGRAMMATION DU MANIPULATEUR DE BLOCS DE DONNEES.

Les routines sont agencées en fonction du niveau auquel elles appartiennent : ainsi, toutes les fonctions de base se retrouvent dans l'ordre où elles ont été décrites au chapitre 2 . Les utilitaires du mot et du byte sont également regroupés, ainsi que toutes les macros utilisant des appels au superviseur. Enfin, la dernière partie concerne tous les messages d'erreurs.

```
      SUBROUTINE BLKACT (BLKID,PRBLK,ITDIM)

C          specification: activer le bloc identifie par blkid,
C          lui assigner une priorite prblk et definir une
C          dimension itdim pour ses items.

           IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX
           COMMON /BMBM3/INDIC
           IF (INDIC.EQ.1) GOTO 100

C          le systeme n'est pas initialise:
C          envoi d'un message d'erreur.

           PRINT 9999
 9999      FORMAT (' Avant de commencer a utiliser le maniPu',
     c     'lateur,',/,' vous devez faire appel a la routine',
     c     ' d''initialisation :SYSINI')
           STOP

C          activation de la "directory" du bloc .

  100      CALL DIRACT (BLKID)

C          decodage de l'identificateur du bloc.

           CALL WIDDCD (BLKID,BLID,NIBLK,FLAG)

C          assignation de la priorite au bloc.

           CALL WOLEPU (BLID,PRBLK)

C          consultation de la premiere page de la table tpb2.
C          examen du contenu du troisieme mot de cette page:
C          ce mot contient la dimension des items du bloc.

           CALL WORIGE (BLID,ADR2)
           ADR = ADR2 + 2
           CALL WORGET (ADR,DIM)

C          le bloc existait-il deja ou vient-il d'etre cree ?

           IF (DIM.EQ.0) 16,17

C          si le bloc vient d'etre cree [c'est-a-dire dim = 0],
C          on lui assigne la dimension itdim.

    16     CALL WORPUT (ADR,ITDIM)
    19     ADR = BLID + 2
           CALL WORPUT (ADR,ITDIM)
           GOTO 20

C          sinon, on verifie si la dimension communiquee
C          [itdim] est la meme que celle donnee a la creation
C          du bloc [dim].

    17     IF (ITDIM.NE.DIM) 18,19

C          la dimension est incorrecte;
C          envoi d'un message d'erreur.

    18     NB = 1
           CALL BMERIT (NB,BLKID)
    20     RETURN
           END



      SUBROUTINE BLKCOP (BLKID1,BLKID2)

C          specification: copier le contenu du bloc identifie par
C          blkid1 dans le bloc identifie par blkid2.
```

```
      IMPLICIT INTEGER (A-U)
       COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c  NBPGMV,PRIMAX
      COMMON /BMBM3/INDIC
       IF (INDIC.EQ.1) GOTO 200

C       le systeme n'est pas initialise;
C       envoi d'un message d'erreur.

       PRINT 9999
 9999  FORMAT (' Avant de commencer a utiliser le manipu',
     c  'lateur,',/,' vous devez faire appel a la routine',
     c  ' d''initialisation :SYSINI')
      STOP

C       decodage de l'identificateur du premier bloc.

  200  CALL WIDDCD (BLKID1,BLID1,NIBLK1,FLAG)
      IF (FLAG.EQ.0) 31,32

C       le premier bloc est desactive;
C       envoi d'un message d'erreur.

   31  NB = 2
       CALL BMERBL (NB,BLKID1)
       RETURN

C       decodage de l'identificateur du deuxieme bloc.

   32  CALL WIDDCD (BLKID2,BLID2,NIBLK2,FLAG)
      IF (FLAG.EQ.0) 33,34

C       le deuxieme bloc est desactive;
C       envoi d'un message d'erreur.

   33  NB = 2
       CALL BMERBL (NB,BLKID2)
       RETURN

C       calcul du nombre de pages de la "directory"
C       du premier bloc.

   34  CALL DIRDIM (BLID1,NDIPG1)

C       calcul du nombre de pages de la "directory"
C       du deuxieme bloc.

       CALL DIRDIM (BLID2,NDIPG2)

C       mise a jour du nombre d'items du deuxieme bloc et
C       et ce en modifiant son identificateur.

      CALL WIDCOD (BLKID2,BLID2,NIBLK1,FLAG)
       ADR = BLID1 + 2

C       recherche de la dimension des items du bloc identifie
C       par blkid1.

      CALL WORGET (ADR,ITDIM1)
      ADR = BLID2 + 2

C       recherche de la dimension des items du bloc identifie
C       par blkid2.

      CALL WORGET (ADR,ITDIM2)
      NBITPG = PGDIM / ITDIM1

C       calcul du nombre de pages de donnees du premier bloc.

      NBDAP1 = ((NIBLK1 - 1) /NBITPG) + 1

C       calcul du nombre de pages de donnees qui sont
C       referencees dans la derniere page "directory"
C       du premier bloc.

      LE = NBDAP1 - ((NDIPG1 - 1) * PDIDIM)
```

```
C           calcul du nombre de pages de donnees du deuxieme bloc.

          NBITPG = PGDIM / ITDIM2
          NBDAP2 = ((NIBLK2 - 1) / NBITPG) + 1
           IF (NDIPG1.LT.NDIPG2) 35,36

C           le deuxieme bloc est plus grand que le premier:d'ou
C           suppression des pages "directory" qui sont de trop.

   35      CALL DIRCUT (BLID2,NDIPG1)
   36      IF (NBDAP1.GE.NBDAP2) GOTO 361

C           suppression des pages de donnees qui sont de trop
C           dans le deuxieme bloc.

          CALL DATCUT (BLID2,NBDAP1)
  361      IF (NDIPG1.NE.1) 37,38
   37      N = NDIPG1 - 1

C           copie des pages de donnees du premier bloc sur
C           celles du second.

           DO  39  I = 1,N
                   CALL DATCOP (BLID1,BLID2,I,PDIDIM)
   39      CONTINUE

C           copie des pages de donnees du premier bloc qui sont
C           referencees dans la derniere page "directory" de ce
C           bloc.

   38      CALL DATCOP (BLID1,BLID2,NDIPG1,LE)
           RETURN
           END



       SUBROUTINE BLKCUT (BLKID,ITEM)

C           specification: tronquer le bloc identifie par blkid
C           a partir de l'item no item [non compris].

          IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX
          COMMON /BMBM3/INDIC
           IF (INDIC.EQ.1) GOTO 300

C           le systeme n'est pas initialise:
C           envoi d'un message d'erreur.

           PRINT 9999
 9999      FORMAT (' Avant de commencer a utiliser le maniPu',
     c     'lateur,',/,' vous devez faire appel a la routine',
     c     ' d''initialisation :SYSINI')
          STOP


C           decodage de l'identificateur du bloc.

  300      CALL WIDDCD (BLKID,BLID,NIBLK,FLAG)
           IF (FLAG.EQ.0) 21,22

C           le bloc est  desactive: envoi d'un message d'erreur.

   21      NB = 3
           CALL BMERBL (NB,BLKID)
          RETURN
   22      IF ((ITEM.LE.NIBLK).AND.(ITEM.GE.0)) GOTO 23

C           le numero de l'item n'existe pas:
C           envoi d'un message d'erreur.

          NB = 4
           CALL BMERIT (NB,BLKID)
```

```
C           mise a jour du nombre d'items du bloc.
   23       CALL WIDCOD (BLKID,BLID,ITEM,FLAG)
C           recherche de la dimension des items du bloc.
           ADR = BLID + 2
           CALL WORGET (ADR,ITDIM)
C           calcul de la page logique contenant l'item no item.
           NBITPG = PGDIM / ITDIM
           DATPG = ((ITEM - 1) / NBITPG) + 1
C           suppression des pages de donnees suivant la page
C           logique no datpg qui contient l'item no item.
           CALL DATCUT (BLID,DATPG)
C           calcul du numero de la page logique "directory"
C           qui reference la page contenant l'item no item.
           DIRPG = ((DATPG - 1) / PDIDIM) + 1
C           suppression des pages "directory" qui suivent
C           la page "directory" no dirpg.
           CALL DIRCUT (BLID,DIRPG)
           RETURN
           END


         SUBROUTINE BLKDEA (BLKID)
C           specification: desactiver le bloc identifie par blkid.
          IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
       c   NBPGMV,PRIMAX
           COMMON /BMBM3/INDIC
           IF (INDIC.EQ.1) GOTO 400
C           le systeme n'est pas initialise:
C           envoi d'un message d'erreur.
           PRINT 9999
 9999      FORMAT (' Avant de commencer a utiliser le manipu',
       c   'lateur,',/,' vous devez faire appel a la routine',
       c   ' d''initialisation :SYSINI')
          STOP
C           desactivation des donnees du bloc.
  400      CALL DATDEA (BLKID)
C           desactivation de la "directory" du bloc et
C           modification de l'identificateur.
          CALL DIRDEA (BLKID)
           RETURN
          END


         SUBROUTINE FILCRE (FINAME,NBFIBL,BLKLST)
C           specification: creer un fichier identifie par le nom
C           contenu dans le vecteur finame et destine a contenir
C           nbfibl blocs identifies par les noms de la liste blklst.
           IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
```

```
     c     NBPGMV,PRIMAX
           COMMON /BMBM3/INDIC
            DIMENSION FINAME (3),BLKLST (NBFIBL)
            IF (INDIC.EQ.1) GOTO 500

C             le systeme n'est pas initialise:
C             envoi d'un message d'erreur.

            PRINT 9999
 9999       FORMAT (' Avant de commencer a utiliser le manipu',
     c      'lateur,',/,' vous devez faire appel a la routine',
     c      ' d''initialisation :SYSINI')
           STOP

C             creation d'un numero d'identification interne pour le
C             fichier ainsi qu'un numero d'identification logique [tf]
C             donnant acces a une entree de la table des fichiers.

  500      FLAG = 1
           CALL FILINI (FINAME,TF,FLAG)

C             ouverture de ce fichier.

           CALL FILACT (TF)

C             creation de l'entete de ce fichier.

           CALL NFISET (TF,NBFIBL)

C             fermeture de ce fichier.

           CALL FILDEA (TF)

C             definition des identificateurs des blocs du fichier
C            ainsi cree.

           FLAG = 0
           DO    302  I = 1 , NBFIBL
                      CALL WIDCOD (BLKLST(I),TF,I,FLAG)
  302      CONTINUE
           RETURN
           END



          SUBROUTINE FILDEF (FINAME,NBFIBL,BLKLST)

C             specification: declarer un fichier qui existe deja,
C             verifier si le nombre effectif de blocs est bien
C             nbfibl et initialiser le contenu des identificateurs
C             des blocs de ce fichier.

           IMPLICIT INTEGER (A-U)
            COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c      NBPGMV,PRIMAX
           COMMON /BMBM3/INDIC
           DIMENSION FINAME (3),BLKLST (NBFIBL)
            IF (INDIC.EQ.1) GOTO 600
            PRINT 9999
 9999       FORMAT (' Avant d''utiliser les routines du manipu',
     c      'lateur',/,' vous devez faire appel a la routine',
     c      ' d''initialisation du systeme.')
           STOP
  600      FLAG = 0

C             assigner une entree de la table des fichiers au
C             fichier qui est declare et lui donner un numero
C             d'identification interne .

           CALL FILINI (FINAME,TF,FLAG)

C             ouverture du fichier.

           CALL FILACT (TF)
```

```
C          verification de la validite du nombre de blocs
C         c'est-a-dire l'argument [nbfibl].

          CALL OFISET (TF,NBFIBL)

C          fermeture du fichier.

          CALL FILDEA (TF)

C          definition des identificateurs des blocs du fichier.

          DO   301  I = 1 , NBFIBL
                    CALL WIDCOD (BLKLST(I),TF,I,FLAG)
  301     CONTINUE
          RETURN
          END



        SUBROUTINE ITEMRD (BLKID,ITEM,VAR)

C          specification:
C          lire l'item [dont le numero est donne par item]
C          du bloc identifie par blkid et le mettre dans le
C          vecteur var.

         IMPLICIT INTEGER (A-U)
          COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
       c   NBPGMV,PRIMAX
         COMMON /BMBM3/INDIC
         DIMENSION VAR (1)

C          decodage de l'identificateur du bloc .

    700   CALL WIDDCD (BLKID,BLID,NIBLK,FLAG)
         IF (FLAG.EQ.0) 1,2

C          le bloc est non active: envoi d'un message d'erreur.

      1   NB = 5
          CALL BMERBL (NB,BLKID)
         RETURN
      2   IF (ITEM.GT.NIBLK) 3,4

C          le no de l'item n'existe pas;
C         envoi d'un message d'erreur.

      3   NB = 2
         CALL BMERIT (NB,BLKID)
          RETURN

C          recherche de la dimension des items du bloc.

      4   ADR = BLID + 2
         CALL WORGET (ADR,ITDIM)

C          calcul du numero logique de la page contenant
C         l'item.

          NBITPG = PGDIM / ITDIM
          DATPG = ((ITEM - 1) / NBITPG) + 1

C          localisation de l'item dans la page.

          DE = ITEM - ((DATPG - 1) * NBITPG)
         DE = (DE - 1) * ITDIM

C          calcul de l'adresse de la page virtuelle contenant l'item.

          CALL DATADR (BLID,DATPG,ADR)
          ADR = ADR - 1 + DE

C          transfert de l'item dans le vecteur var.

         DO  5  I = 1,ITDIM
```

```
            ADR = ADR + 1
            CALL WORGET (ADR,VAR (I))
 5      CONTINUE
       RETURN
        END



      SUBROUTINE ITEMUR (BLKID,ITEM,VAR)

C          specification: modifier ou creer l'item [dont le
C          numero est donne par item] avec le contenu du
C          vecteur var.

       IMPLICIT INTEGER (A-U)
        COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c   NBPGMV,PRIMAX
       COMMON /BMBM3/INDIC
        DIMENSION VAR (1)

C          decodage de l'identificateur du bloc.

        CALL WIDDCD (BLKID,BLID,NIBLK,FLAG)
        IF (FLAG.EQ.0) 11,12

C          le bloc est non active: envoi d'un message d'erreur.

 11     NB = 6
        CALL BMERBL (NB,BLKID)

 12     IF ((NIBLK + 1).LT.ITEM) 13,14

C          le no de l'item est incorrect;
C          envoi d'un message d'erreur.

 13     NB = 3
        CALL BMERIT (NB,BLKID)


C          si l'on cree un item, le nombre d'items du bloc
C          est modifie.

 14     IF ((NIBLK + 1).EQ.ITEM)
     c  CALL WIDCOD (BLKID,BLID,ITEM,FLAG)

C          recherche de la dimension des items du bloc.

        ADR = BLID + 2
        CALL WORGET (ADR,ITDIM)

C          calcul du numero logique de la page contenant l'item.

        NBITPG = PGDIM / ITDIM
        DATPG = ((ITEM - 1) / NBITPG) + 1
        DE = ITEM - ((DATPG - 1)*NBITPG)

C          localisation de l'item dans la page.

       DE = (DE - 1) * ITDIM

C          calcul de l'adresse de la page virtuelle contenant
C          l'item.

       CALL DATADR (BLID,DATPG,ADR)

C          indiquer que la page a ete modifiee.
C          adc contient l'adresse de l'entree de la carte qui
C          reference la page ainsi modifiee.

       ADC = ADCA + (((ADR - ADMV) / PGDIM) * 2)
        FLAG = 1
        CALL BYFLPU (ADC,FLAG)
       ADR = ADR - 1 + DE

C          transfert du contenu du vecteur var dans l'item.
```

```
      DO  15  I = 1,ITDIM
              ADR = ADR + 1
              CALL WORPUT (ADR,VAR(I))
15    CONTINUE
      RETURN
      END


      SUBROUTINE SYSINI (NUMPG,NUMFIL,PRPG)

C      specification:
C      reserver la zone necessaire au stockage des donnees de
C     l'utilisateur.
C     reserver les zones ou seront implantees la carte et la
C     table des fichiers.
C     reserver deux pages pour le buffer utilise dans les
C     operations d'entree/sortie et pour la derniere page
C     de la table des pages libres d'un fichier.
C     sauver les adresses de ces differentes zones.
C      donner la priorite maximale accordee a une page de
C     donnees.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     C NBPGMV,PRIMAX
      COMMON /BMBM2/ADRBUF,NUMBUF
     COMMON /BMBM3/INDIC
      DATA PGDIM/512/
     NBPGMV = NUMPG
     NBFIUT = NUMFIL
     PRIMAX = PRPG
     INDIC = 1

C      reservation de la zone destinee a recevoir les donnees
C      de l'utilisateur ainsi que le buffer reserve aux ope-
C     rations d'entree/sortie.

     NUMPG = NUMPG + 1
      CALL ALLOCA (NUMPG,ADMV)
     ADRBUF = ADMV + (NBPGMV * PGDIM)
     NUMBUF = ADRBUF / PGDIM

C      reservation de la zone destinee a recevoir la carte.

     CALL MAPCRE (ADCA)

C      reservation de la zone destinee a recevoir
C      la table des fichiers.

      CALL TFICRE (ADTF)

C      reservation de la zone destinee a recevoir le buffer
C      de la table des pages libres d'un fichier.

      CALL TPLFCR (ADTPLF)

C      initialisation du nombre d'entrees dans une page
C     "directory".

     PDIDIM = PGDIM - 3
     RETURN
     END
```

```
      SUBROUTINE ALLOCA (NUMPG,ADR)

C           specification: reserver la zone destinee a recevoir
C           les donnees de l'utilisateur.

          IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX

C           calcul de la taille de cette zone.

           DIM = NUMPG * PGDIM

C           recherche de l'adresse de la partie de la memoire
C           qui est disponible.

          CALL BMGFFA (ADR)
          ADR = ((ADR / PGDIM) + 1) * PGDIM

C           reservation a partir de cette adresse.

          CALL BMGMEM (ADR,DIM,ERCODE)
          IF (ERCODE.EQ.0) GOTO 341

C           la reservation n'a pu se faire;
C           envoi d'un message d'erreur.

          NB = 1
          CALL BMERME (NB,ERCODE)
  341     RETURN
          END



      SUBROUTINE DATDEA (BLKID)

C           specification: desactivation des donnees du bloc
C           identifie par blkid.

          IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX

C           decodage de l'identificateur du bloc.

          CALL WIDDCD (BLKID,BLID,NIBLK,FLAG)
          IF (FLAG.EQ.0) 61,62

C           le bloc est deja desactive;
C           envoi d'un message d'erreur.

   61     NB = 4
          CALL BMERBL (NB,BLKID)
          RETURN

C           decodage de l'entete de la table tpb1.

   62     ADR1 = BLID
   63     CALL WORIGE (ADR1,ADR2)

C           desactivation des pages de donnees referencees dans
C           une page "directory" determinee.

          J = PGDIM - 1
          DO  64  I = 3,J
                  PT1 = ADR1 + I
                  PT2 = ADR2 + I
                  CALL DATOUT (BLID,PT1,PT2)
   64     CONTINUE

C           est-on a la fin de la table tpb1 du bloc ?
```

```
      ADR = ADR1 + 1
      CALL WORGET (ADR,ADR1)
      IF (ADR1.NE.0) GOTO 63
       RETURN
       END



      SUBROUTINE DIRACT (BLKID)

C          specification: activer la "directory" du bloc
C          identifie par blkid.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c  NBPGMV,PRIMAX

C          decodage de l'identificateur du bloc.

      CALL WIDDCD (BLKID,TF,BLK,FLAG)
      IF (FLAG.EQ.1) 41,42

C          le bloc est deja active: envoi d'un message d'erreur.

   41   NB = 1
      CALL BMERBL (NB,BLKID)
      RETURN
   42   IF (PRBLK.LE.PRIMAX) GOTO 45
      NB = 7
      CALL BMERME (NB,BLKID)
      RETURN

C          fp designe la page "fichier" contenant la premiere
C          page de la table tpb2 du bloc.

   45   FP = BLK

C          allocation de deux pages virtuelles et
C          mise a jour de la carte.

      PRPG = PRIMAX + 1
      INT = 0
      CALL PGMGET (ADR1)
     ADC1 = ADCA + (((ADR1 - ADMV) / PGDIM) * 2)
      CALL PGMSMP (ADC1,INT,PRPG,INT,INT,INT,INT)
      CALL PGMGET (ADR2)
     ADC2 = ADCA + (((ADR2 - ADMV) / PGDIM) * 2)
      CALL PGMSMP (ADC2,INT,PRPG,INT,INT,INT,INT)

C          ouverture [si necessaire] du fichier contenant le
C          bloc et calcul de l'adresse de l'entree de la table
C          des fichiers qui correspond au fichier contenant le
C          le bloc.

      ADR = ADTF + ((TF - 1) * 4) + 3
      CALL WOLEGE (ADR,SEM)

C          la variable sem indique le nombre de blocs actifs du
C          fichier; si elle vaut 0 ,cela signifie donc qu'aucun
C          bloc n'est actif et que le fichier les contenant est
C          est ferme.

      IF (SEM.EQ.0) 43,44

C          ouverture du fichier no tf.

   43   CALL FILACT (TF)
   44   SEM = SEM + 1
      CALL WOLEPU (ADR,SEM)

C          lecture de la premiere page "directory" "fichier"
C          du bloc.

      CALL PGFIRD (TF,FP,ADR2)
```

```
C          ecriture de l'entete des tables tpb1 et tpb2 et
C          modification de l'identificateur du bloc.

         FLAG = 1
          CALL WORIPU (ADR1,ADR2)
          ADR = ADR1 + 1
          INT = 0
          CALL WORPUT (ADR,INT)
          CALL WORCOD (ADR2,TF,BLK)
         ADR2 = ADR2 + 1
          CALL WORIGE (ADR2,NIBLK)
          CALL WIDCOD (BLKID,ADR1,NIBLK,FLAG)
          RETURN
          END



       SUBROUTINE DIRDEA (BLKID)

C          specification: desactivation de la "directory"
C          du bloc identifie par blkid.

          IMPLICIT INTEGER (A-U)
          COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     C    NBPGMV,PRIMAX
          COMMON /BMBM2/ ADRBUF,NUMBUF

C          decodage de l'identificateur du bloc.

          CALL WIDDCD (BLKID,BLID,NIBLK,FLAG)

C          decodage de l'entete de la table tpb1.

          CALL WORIGE (BLID,ADR2)

C          decodage de l'entete de la table tpb2.

          CALL WORDCD (ADR2,TF,BLK)
          ADR = ADR2 + 1
          CALL WORIGE (ADR,INT)

C          mise a jour [si necessaire] du nombre d'items du bloc.

          IF (INT.NE.NIBLK) 51,52
   51     CALL WORIPU (ADR,NIBLK)

C          fp designe la premiere page "fichier" de la table tpb2
C          du bloc.

   52     FP = BLK

C          ecriture sur fichier de la premiere page
C          de la table tpb2 du bloc.

         CALL PGCOPY (ADR2,ADRBUF)
          CALL PGFIWR (TF,FP,ADRBUF)
          ADR1 = BLID

C          ecriture sur fichier de la page suivante
C          de cette table.

   53     CALL DIROUT (ADR1,ADR2)
          IF (ADR1.EQ.0) 54,53

C          on se trouve a la fin de la table tpb2.

   54     FLAG = 0

C          modification de l'identificateur du bloc.

          CALL WIDCOD (BLKID,TF,BLK,FLAG)

C          fermeture [si necessaire] du fichier contenant le
C          bloc et calcul de l'entree de la table des fichiers
C          correspondant au fichier contenant le bloc.
```

```
      ADR = ADTF + ((TF - 1) * 4) + 3
      CALL WOLEGE (ADR,SEM)

C       la variable sem indique le nombre de blocs du fichier
C       qui sont encore actifs : si elle vaut 1 , cela signifie
C       donc que le seul bloc actif est celui qu'on veut
C       precisement desactiver.

      IF (SEM.EQ.1) 55,56

C       fermeture du fichier no tf.

   55 CALL FILDEA (TF)
   56 SEM = SEM - 1
      CALL WOLEPU (ADR,SEM)
      RETURN
      END



      SUBROUTINE FILACT (TF)

C       specification: ouvrir le fichier no tf, [tf] designant
C       un numero d'identification logique.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     C  NBPGMV,PRIMAX
      DIMENSION FINAME (3)

C       calcul de l'endroit dans la table des fichiers ou se
C       trouve le le numero d'identification interne [jfn] de
C       ce fichier.

      ADR = ADTF + ((TF - 1) * 4) + 3
      CALL WORIGE (ADR,JFN)
      TYPE 1000,JFN
 1000 FORMAT (' JFN VAUT:',I7)

C       ouverture du fichier.

      CALL BMOPEN (JFN,ERCODE)
      IF (ERCODE.EQ.0) GOTO 321

C       l'ouverture ne s'est pas effectuee correctement :
C       envoi d'un message d'erreur specifiant le nom du
C       fichier.

      NB = 1
      ADR = ADR - 4
      DO  322  I = 1 , 3
               ADR = ADR + 1
               CALL WORGET (ADR,FINAME(I))
  322 CONTINUE
      CALL BMERFI (NB,FINAME,ERCODE)
  321 RETURN
      END



      SUBROUTINE FILDEA (TF)

C       specification: fermeture du fichier no tf, [tf]
C       designant un numero d'identification logique.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     C  NBPGMV,PRIMAX
      DIMENSION FINAME (1)

C       le nombre actuel de pages du fichier est consigne
C       dans la derniere page de la table tplf [page no 1
C       du fichier] et l'on verifie donc si la derniere
```

```
C          page de cette table se trouve encore en memoire
C          virtuelle et dans ce cas, il faut alors recopier
C         cette page sur le fichier.

          CALL WLLGET (ADTPLF,INT)
          IF (INT.NE.TF) GOTO 332
          FP = 0
          CALL PGFIWR (TF,FP,ADTPLF)
          INT = 0
          CALL WLLPUT (ADTPLF,INT)

C          calcul de l'adresse de l'entree de la table des
C          fichiers correspondant au fichier en question.

  332     ADR = ADTF + ((TF - 1) * 4) + 3

C          recherche du job file number du fichier.

          CALL WORIGE (ADR,JFN)

C          fermeture du fichier.

          CALL BMCLOS (JFN,ERCODE)
          IF (ERCODE.EQ.0) GOTO 331

C          la fermeture ne s'est pas effectuee correctement
C          envoi d'un message d'erreur specifiant le nom du
C         fichier.

          NB = 2
         ADR = ADR - 4
         DO  333  I = 1 , 3
                  ADR = ADR + 1
                  CALL WORGET (ADR,FINAME(I))
  333     CONTINUE
          CALL BMERFI (NB,FINAME,ERCODE)
  331     RETURN
          END



        SUBROUTINE FILINI (FINAME,TF,FLAG)

C          specification: assigner une entree de la table des
C          fichiers au fichier identifie par finame et lui
C          donner un numero d'identification interne; tf est un
C          un output qui designe une entree de la table des
C          fichiers et le parametre flag suivant qu'il vaut
C          0 ou 1 indique que le fichier existe deja ou va
C         etre cree.

          IMPLICIT INTEGER (A-U)
          COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
      C    NBPGMV,PRIMAX
          DIMENSION FINAME (3)
         DIMENSION STRING (3)

C          recherche d'une entree libre dans la table des
C         fichiers.

          ADR = ADTF - 4
          DO  191    I = 1 , NBFIUT
                     ADR = ADR + 4
                     CALL WORGET (ADR,INT)
                     TF = I
                     IF (INT.EQ.0) GOTO 192
  191     CONTINUE

C          calcul de la longueur du nom du fichier.

  192     L = LENGTH (FINAME)
          AD = ADR

C          transformation du nom du fichier en code asciz.
```

```
      CALL BMASCI (FINAME,L,STRING,ERCODE)
      IF (ERCODE.EQ.0) GOTO 193

C      il s'est passe quelque chose d'anormal durant
C      la transformation: envoi d'un message d'erreur.

      NB = 3
      CALL BMERFI (NB,FINAME,ERCODE)
      RETURN

C      assignation d'un numero d'identification interne
C      au fichier.

  193 CALL BMGJFN (STRING,JFN,FLAG,ERCODE)
      IF (ERCODE.EQ.0) GOTO 194

C      quelque chose d'anormal s'est produit.
C      envoi d'un message d'erreur.

      NB = 4
      CALL BMERFI (NB,FINAME,ERCODE)
  194 ADR = AD - 1
      DO  195  I = 1 , 3
               ADR = ADR + 1
               CALL WORPUT (ADR,STRING(I))
  195 CONTINUE

C      sauvetage du numero d'identification interne
C      du fichier dans la table des fichiers.

      AD = AD + 3
      INT = 9
      CALL WORCOD (AD,INT,JFN)
      RETURN
      END



      SUBROUTINE MAPCRE (ADR)

C      specification:
C      reservation de la zone destinee a recevoir la carte.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c NBPGMV,PRIMAX

C      recherche de l'adresse de la memoire disponible.

      CALL BMGFFA (ADR)

C      calcul de la taille de la carte.

      DIM = NBPGMV * 2

C      reservation de memoire pour la carte.

      CALL BMGMEM (ADR,DIM,ERCODE)
      IF (ERCODE.EQ.0) GOTO 351

C      la reservation n'a pu se faire:
C      envoi d'un message d'erreur.

      NB = 2
      CALL BMERME (NB,ERCODE)
  351 RETURN
      END



      SUBROUTINE NFISET (TF,NBFIBL)

C      specification: creation de l'entete du nouveau
C      fichier identifie par le numero logique tf et
```

```
C          destine a contenir nbfibl blocs.

           IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX

C          initialisation du nombre de pages du fichier.

           NBFIPG = NBFIBL + 1

C          initialisation de la table tplf et transfert
C          sur fichier.

           CALL WLLGET (ADTPLF,INT)
           FP = 0
           IF (INT.NE.0) CALL PGFIWR (INT,FP,ADTPLF)
           INT = 0
           CALL WORPUT (ADTPLF,INT)
           CALL WLLPUT (ADTPLF,TF)
           ADR = ADTPLF + 1
           CALL WORCOD (ADR,NBFIBL,NBFIPG)
           CALL PGFIWR (TF,FP,ADTPLF)
           CALL WORPUT (ADTPLF,INT)

C          initialisation des tables des pages des blocs
C          et ensuite transfert sur fichier.

           ADR = ADMV
           AD = ADR + 1
           FP = 1
           DO  392  I = 1 , NBFIBL
                    CALL WORCOD (ADR,TF,I)
                    CALL WORPUT (AD,INT)
                    CALL PGFIWR (TF,FP,ADR)
                   FP = FP + 1
 392       CONTINUE
           RETURN
           END



          SUBROUTINE OFISET (TF,NBFIBL)

C          specification: le fichier no tf existe deja et le
C          nombre de blocs de ce fichier doit coincider avec
C          le nombre de blocs communique par l'utilisateur [nbfibl].

           IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX
          DIMENSION FINAME (3)

C          la fenetre de la table tplf est-elle occupee ?
C          si oui ,elle est videe de son contenu sur le fichier
C          auquel elle correspond.

           CALL WLLGET (ADTPLF,INT)
           IF (INT.EQ.0) GOTO 381
           FP = 0
           CALL PGFIWR (INT,FP,ADTPLF)

C          transfert ,dans la fenetre, de la derniere page
C          de la table tplf du fichier no tf.

 381       CALL PGFIRD (TF,FP,ADTPLF)
           CALL WOLEGE (ADTPLF + 1,INT)
           IF (INT.EQ.NBFIBL) GOTO 382

C          le nombre de blocs communiques est incorrect:
C          envoi d'un message d'erreur.

           NB = 5
           ADR = ADTF + ((TF - 1) * 4) - 1
          DO  383  I = 1 , 3
                   ADR = ADR + 1
```

```
                CALL WORGET (ADR,FINAME(I))
  383      CONTINUE
           CALL BMERFI (NB,FINAME,INT)
  382      RETURN
           END



        SUBROUTINE TFICRE (ADR)

C            specification: reservation de la zone destinee
C            a recevoir la table des fichiers.

           IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX

C            recherche de l'adresse de la memoire disponible.

           CALL BMGFFA (ADR)

C            calcul de la taille de la table des fichiers.

          DIM = NBFIUT * 4

C            reservation effective de la memoire.

           CALL BMGMEM (ADR,DIM,ERCODE)
           IF (ERCODE.EQ.0) GOTO 361

C            la reservation ne s'est pas effectuee:
C            envoi d'un message d'erreur.

           NB = 3
           CALL BMERME (NB,ERCODE)
  361      RETURN
           END



        SUBROUTINE TPLFCR (ADR)

C            specification: reservation du buffer destine  a
C            recevoir la table des pages libres d'un fichier.

           IMPLICIT INTEGER (A-U)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX

C            recherche de l'adresse de la memoire disponible.

           CALL BMGFFA (ADR)
           ADR = ((ADR /PGDIM) + 1) * PGDIM

C            reservation effective de la memoire.

           CALL BMGMEM (ADR,PGDIM,ERCODE)
           IF (ERCODE.EQ.0) GOTO 371

C            la reservation n'a pu se faire:
C            envoi d'un message d'erreur.

           NB = 4
           CALL BMERME (NB,ERCODE)
  371      RETURN
           END

 MEM2.BM.60

        SUBROUTINE DATADR (BLID,DATPG,ADR)

C            specification: calcul de l'adresse virtuelle de la
C            page logique de donnees [no datpg] du bloc pointe
C            par blid.
```

```
      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c  NBPGMV,PRIMAX

C      calcul du numero de la page "directory" ou est
C      referencee la page logique de donnees.

      DIRPG = ((DATPG - 1) / PDIDIM) + 1

C      localisation a l'interieur de la page "directory".

      DE = DATPG - ((DIRPG - 1) * PDIDIM) + 2

C      calcul de l'adresse virtuelle de la page "directory".

      CALL DIRADR (BLID,DIRPG,ADR1)

C      decodage de l'entete de la page "directory"
C      de la table trb1.

      CALL WORIGE (ADR1,ADR2)

C      pt1 et pt2 designent les adresses des entrees des
C      tables trb1 et trb2 correspondant a la page logique
C      no datpg.

      PT1 = ADR1 + DE
      PT2 = ADR2 + DE


C      recherche de la page virtuelle contenant la page
C      de donnees.

      CALL DATIN (BLID,PT1,PT2)

C      sauvetage de l'adresse de cette page dans
C      la variable adr.

      CALL WORGET (PT1,ADR)
      RETURN
      END



  SUBROUTINE DATCOP (BLID1,BLID2,DIRPG,NBDAPG)

C       specification: copier les "nbdapg" pages referencees
C       dans la page "directory" [no dirpg] du bloc pointe
C       par blid1 dans les pages correspondantes du bloc
C       pointe par blid2.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c  NBPGMV,PRIMAX
      FLAG = 1
      N = NBDAPG + 1

C       calcul de l'adresse de la page "directory" no dirpg du
C       bloc pointe par blid1.

      CALL DIRADR (BLID1,DIRPG,AD1)

C       idem pour l'autre bloc.

      CALL DIRADR (BLID2,DIRPG,AD2)

C       decodage de l'entete de la page "directory" pour
C       le premier bloc.

      CALL WORIGE (AD1,AD12)

C       idem pour le second bloc.
```

```
      CALL WORIGE (AD2,AD22)
C     copie des pages.
      DO  81  I = 3,N
C     recherche d'une page de donnees du premier bloc.
            CALL DATIN (BLID1,AD1 + I,AD12 + I)
C     recherche de la page de meme numero dans
C     le second bloc.
            CALL DATIN (BLID2,AD2 + I,AD22 + I)
C     sauvetage de l'adresse de la premiere page dans ada.
            CALL WORGET (AD1 + I,ADA)
C     sauvetage de l'adresse de la seconde page dans adb.
            CALL WORGET (AD2 + I,ADB)
C     calcul de l'entree de la carte correspondant a la
C     la page virtuelle d'adresse adb.
           ADC = ADCA + (((ADB - ADMV) / PGDIM) * 2)
C     indication du fait que la page virtuelle d'adresse adb
C     a ete modifiee.
            CALL BYFLPU (ADC,FLAG)
C     copie du contenu de la page virtuelle d'adresse ada
C     dans la page virtuelle d'adresse adb.
            CALL PGCOPY (ADA,ADB)
   81 CONTINUE
      RETURN
      END


      SUBROUTINE DATCUT (BLID,DATPG)
C     specification: detruire les pages de donnees du bloc
C     pointe par blid et ce a partir de la page dont le
C     numero est (datpg + 1).
      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     C NBPGMV,PRIMAX
C     calcul du numero de la page "directory" ou est
C     referencee la page logique no datpg.
      DIRPG = ((DATPG - 1) / PDIDIM) + 1
      DE = DATPG - ((DIRPG - 1) * PDIDIM) + 1
C     si la variable "de" est <= a pdidim, alors cela signi-
C     fie que l'entree no "de" de la page "directory" refe-
C     rence la premiere page de donnees qui doit etre de-
C     truite; sinon, la premiere page de donnees qui doit
C     etre detruite est referencee dans la premiere entree
C     de la page "directory" suivante.
      IF (DE.LE.PDIDIM) 71,72
C     calcul de l'adresse de la page "directory" no dirpg.
   71 CALL DIRADR (BLID,DIRPG,ADR1)
C     decodage de l'entete de cette page "directory".
      CALL WORIGE (ADR1,ADR2)
```

```
      INT = 0
      ADR1 = ADR1 + 2
      ADR2 = ADR2 + 2
      DO  73  I = DE,PDIDIM
              PT1 = ADR1 + I
              PT2 = ADR2 + I

C      destruction d'une page logique de donnees.

            CALL DATDS (BLID,PT1,PT2)
             CALL WORPUT (PT1,INT)
             CALL WORPUT (PT2,INT)
73    CONTINUE

C      calcul du nombre de pages de la "directory" du bloc.

72    CALL DIRDIM (BLID,NBDIPG)
      IF (NBDIPG.GT.DIRPG) 74,75

C      cas ou la page "directory" no dirpg n'est pas la
C      derniere page de la "directory" du bloc.

74    N = DIRPG + 1

C      destruction des pages de donnees referencees dans les
C      pages "directory" qui suivent.

      DO  76  I = N,NBDIPG
              CALL DIRADR (BLID,I,ADR1)
              CALL WORIGE (ADR1,ADR2)
              ADR1 = ADR1 + 2
              ADR2 = ADR2 + 2
              DO  77  J = 1,PDIDIM
                      PT1 = ADR1 + J
                      PT2 = ADR2 + J
                      CALL DATDS (BLID,PT1,PT2)
77            CONTINUE
76    CONTINUE
75    RETURN
      END



    SUBROUTINE DIRADR (BLID,DIRPG,ADR1)

C       specification: calcul de l'adresse de la page
C       "directory" [no dirpg] de la table trbi du bloc
C      pointe par blid.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c  NBPGMV,PRIMAX
      ADR1 = BLID
      IF (DIRPG.EQ.1) RETURN

C       parcours de la chaine de la table trbi.

      N = DIRPG - 1
      DO  91  K = 1,N
              ADR = ADR1 + 1
              CALL WORGET (ADR,FADR1)
              IF (FADR1.NE.0) 92,93
92            ADR1 = FADR1
              GOTO 91
93            CALL WORIGE (ADR1,ADR2)
             type 94,admv,adr1,adr2
94            format (' admv ',i7,' adr1 ',i7,' adr2 ',i7)

C       recherche des pages "directory" des tables trb1
C       et trb2 qui suivent celles d'adresses respectives
C      adr1 et adr2.

              CALL DIRIN (ADR1,ADR2)
           type 95,adr1,adr2
95          format (' adr1 apres dirin',i7,' adr2 apres dirin',i7)
```

```
 91     CONTINUE
        RETURN
        END


      SUBROUTINE DIRCUT (BLID,DIRPG)

C          specification: detruire les pages "directory" du
C          bloc pointe par blid suivant la page "directory"
C         no dirpg.

          IMPLICIT INTEGER (A-U)
          COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c    NBPGMV,PRIMAX

C          calcul de la taille de la "directory" du bloc.

          CALL DIRDIM (BLID,NBDIPG)
          IF (NBDIPG.NE.DIRPG) 101,102

C          calcul de l'adresse de la page "directory" no dirpg.

 101      CALL DIRADR (BLID,DIRPG,ADR1)

C          decodage de l'entete de cette page.

          CALL WORIGE (ADR1,ADR2)
          AD = ADR2 + 1
          ADR = ADR1 + 1
          CALL WORGET (ADR,FADR1)
          IF (FADR1.EQ.0) 103,104

C          recherche de la page "directory" qui suit celle dont
C          le numero est donne par dirpg.

 103      CALL DIRIN (ADR1,ADR2)
          GOTO 105
 104      INT = 0
          CALL WORPUT (ADR,INT)
          CALL WORIGE (FADR1,FADR2)
          ADR1 = FADR1
          ADR2 = FADR2
 105      FFP = 0
          CALL WOLEPU (AD,FFP)
          N = NBDIPG - DIRPG

C          troncature de la "directory".

          DO  106  I = 1,N

C          destruction des pages "directory" d'adresses
C          adr1 et adr2 ainsi que de la version "fichier".

                    CALL DIRDS (ADR1,ADR2)
 106      CONTINUE
 102      RETURN
          END


      SUBROUTINE DIRDIM (BLID,NBDIPG)

C          specification: calcul de la taille de la "directory"
C          (exprimee en pages) du bloc identifie par blid.

          IMPLICIT INTEGER (A-U)
          COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c    NBPGMV,PRIMAX

C          decodage de l'entete de la premiere page de la
C          table trb1 du bloc identifie par blid.

          CALL WORIGE (BLID,ADR2)
```

```
      ADR2 = ADR2 + 1

C      decodage de l'entete de la premiere page de la table
C      tab2 afin de connaitre le nombre d'items du bloc et
C      la taille de ces items.

      CALL WORIGE (ADR2,NIBLK)
      ADR2 = ADR2 + 1
      CALL WORGET (ADR2,ITDIM)
      NBITPG = PGDIM / ITDIM
      NBDAPG = ((NIBLK - 1) / NBITPG) + 1
      NBDIPG = ((NBDAPG - 1) / PDIDIM) + 1
      RETURN
      END


      SUBROUTINE DATDS (BLID,PT1,PT2)

C      specification: destruction d'une page de donnees:
C      destruction, si elle existe, de la version en memoire
C      virtuelle de la page de donnees [l'adresse est alors
C      contenue dans le mot pointe par pt1] et
C      destruction de sa version "fichier" [l'adresse est
C      alors contenue dans le mot pointe par pt2].

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c  NBPGMV,PRIMAX

C      adr est l'adresse virtuelle de la page de donnees
C      si elle existe en memoire virtuelle.

      CALL WORGET (PT1,ADR)

C      fp est l'adresse "fichier" de la page de donnees

      CALL WORGET (PT2,FP)

C      desallocation  de la page "fichier" occupee par la
C      page de donnees.

      IF (FP.EQ.0) GOTO 107
       CALL WORIGE (BLID,ADR2)
       CALL WOLEGE (ADR2,TF)
      CALL PGFFRE (TF,FP)

C      desallocation de la page virtuelle occupee par la
C      page de donnees.

      IF (ADR.NE.0) CALL PGMFRE (ADR)
  107 RETURN
      END


      SUBROUTINE DATIN (BLID,PT1,PT2)

C      specification:
C      referenciation d'une page de donnees:
C      si elle existe sur fichier et que sa version virtuelle
C       existe deja, son adresse virtuelle est contenue dans
C      le mot pointe par pt1,
C      si elle existe seulement sur fichier, son numero est
C      consigne dans le mot pointe par pt2 et elle est alors
C      transferee en memoire virtuelle et son adresse virtu-
C      elle est sauvee dans le mot pointe par pt1,
C      si n'existe pas encore, une page "fichier" et une pa-
C      ge virtuelle sont allouees et leurs adresse et numero
C      sont respectivement sauves dans les mots pointes par
C      pt1 et pt2.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c  NBPGMV,PRIMAX
```

```
C          int represente la reference a la page virtuelle ou
C          se trouve eventuellement la page de donnees.

           CALL WORGET (PT1,INT)
           IF (INT.EQ.0) 141,142

C          cas ou la reference est nulle, c'est-a-dire que la
C          page de donnees n'est pas encore en memoire virtuelle.
C          decodage de l'entete de la premiere page "directory"
C          de la table tpb1.

  141      CALL WORDCD (BLID,PRBLK,ADR2)

C          decodage de la premiere page de la table tpb2.

           CALL WOLEGE (ADR2,TF)

C          fp represente la reference a la page "fichier"
C          eventuellement occupee par la page de donnees
C          [si cette page de donnees existe deja].

           CALL WORGET (PT2,FP)
           INDIC = 0
           IF (FP.EQ.0) 143,144

C          cas ou la page de donnees n'existe pas encore.
C          allocation d'une page "fichier" pour la page
C          de donnees que l'on cree.

  143      CALL PGFGET (TF,FP)

C          on note le numero de cette page dans l'entree
C          d'adresse pt2 de la table tpb2.

           CALL WORPUT (PT2,FP)
           INDIC = 1
  144      FLAG = 0

C          allocation d'une page virtuelle
C          pour la page de donnees.

           CALL PGMGET (ADR)
          ADC = ADCA + (((ADR - ADMV) / PGDIM) * 2)
           CALL PGMSMP (ADC,FLAG,PRIMAX,PRBLK,TF,FP,PT1)

C          indic = 0 signifie que la page de donnees existe
C          deja mais qu'elle n'est pas encore implantee en
C          memoire virtuelle.
C          dans ce cas, lecture de la page "fichier"
C          qui la contient.

           IF (INDIC.EQ.0) CALL PGFIRD (TF,FP,ADR)

C          sauvetage de l'adresse virtuelle de la page de donnees
C          dans l'entree [d'adresse pt1] de la table tpb1.

           CALL WORPUT (PT1,ADR)
  142      RETURN
           END



         SUBROUTINE DATMAP (ADR)

C          specification: decrementer la priorite des pages
C          virtuelles occupees par des pages de donnees, sauf
C          la page virtuelle d'adresse adr dont la priorite est
C          portee a son maximum.

           IMPLICIT INTEGER (A-U)
           COMMON /BABM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
       C   NBPGMV,PRIMAX
           DO  311  I = 1 , NBPGMV
                    ADC = ADCA + ((I - 1) * 2)
```

```
                CALL BYPBGE (ADC,PRBLK)
                 CALL BYPPGE (ADC,PRPG)
                 IF((PRPG.LE.PRBLK).OR.(PRPG.EQ.(PRIMAX + 1)))
     c           GOTO 311
                 PRPG = PRPG - 1
                 CALL BYPPPU (ADC,PRPG)
311      CONTINUE
        ADC = ADCA + (((ADR - ADMV) / PGDIM) * 2)
         CALL BYPPPU (ADC,PRIMAX)
         RETURN
         END



      SUBROUTINE DATOUT (BLID,PT1,PT2)

C          specification: recopier,si elle existe, la version
C          en memoire virtuelle d'une page de donnees [dont
C          l'adresse est contenue dans le mot pointe par pt1]
C          sur la page "fichier" dont l'adresse est contenue
C        dans le mot pointe par pt2.

         IMPLICIT INTEGER (A-U)
         COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c   NBPGMV,PRIMAX

C          adr est l'adresse virtuelle de la page de donnees
C          [si elle se trouve en memoire virtuelle].

        CALL WORGET (PT1,ADR)
         IF (ADR.NE.0) 151,152

C          cas ou cette page se trouve en memoire virtuelle.
C          on examine si la page a subi des modifications.

   151   ADC = ADCA + (((ADR - ADMV) / PGDIM) * 2)
         CALL BYFLGE (ADC,FLAG)
         IF (FLAG.NE.0) 153,154

C          si oui,recopie sur fichier et desallocation de la
C          page virtuelle occupee.

   153   CALL WORGET (PT2,FP)
         CALL WORIGE (BLID,ADR2)
         CALL WOLEGE (ADR2,TF)
         CALL PGFIWR (TF,FP,ADR)
   154   CALL PGMFRE (ADR)
   152   RETURN
         END



      SUBROUTINE DIRDS (ADR1,ADR2)

C          specification: desallouer les pages virtuelles
C          "directory" [d'adresses adr1 et adr2] des tables
C          tpb1 et tpb2 et desallouer la page "fichier" "direc-
C          tory" correspondant a la page virtuelle "directory"
C          [d'adresse adr2] de la table tpb2.

         IMPLICIT INTEGER (A-U)
         COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c   NBPGMV,PRIMAX
         FADR1 = 0
         IF (ADR1.NE.0) 131,132
   131   ADR = ADR1 + 1
         CALL WORGET (ADR,FADR1)

C          liberation de la page virtuelle occupee par la
C          page "directory" de la table tpb1.

         CALL PGMFRE (ADR1)
   132   CALL WORDCD (ADR2,TF,BLK)
         ADR = ADR2 + 1
```

```
      CALL WORDCD (ADR,FFP,FP)

C      liberation de la page virtuelle occupee par la
C      page "directory" de la table tpb2.

      CALL PGMFRE (ADR2)

C      liberation de la page "fichier" occupee par la
C      page "directory" de la table tpb2.

      CALL PGFFRE (TF,FP)
      IF (FFP.EQ.0) RETURN

C      la page "directory" qui vient d'etre detruite
C      n'est pas la derniere page de la table des pages.

      IF (FADR1.NE.0) 133,134

C      cas ou l'on ne se trouve pas a la fin de la table tpb1.

  133 CALL WORIGE (FADR1,FADR2)
      GOTO 135

C      cas ou l'on se trouve a la fin de la table tpb1:
C      il suffit alors de transferer en memoire virtuelle
C      la page suivante de la table tpb2.

  134 CALL PGMGET (FADR2)
      PRPG = PRIMAX + 1
      INT = 0

C      calcul de l'adresse de l'entree de la carte corres-
C      pondant a la page virtuelle qui vient d'etre allouee.

     FAD2 = ADCA + (((FADR2 -ADMV) / PGDIM) * 2)

C      ajustement de le carte.

      CALL PGMSMP (FAD2,INT,PRPG,INT,INT,INT,INT)

C      transfert en memoire virtuelle de la page "fichier"
C      "directory" [de la table tpb2] qui suit la page
C      "fichier" qui vient d'etre supprimee.

      CALL PGFIRD (TF,FFP,FADR2)

C      sauvetage des adresses virtuelles des pages suivantes
C      des tables tpb1 et tpb2.

  135 ADR1 = FADR1
      ADR2 = FADR2
      RETURN
      END


     SUBROUTINE DIRIN (ADR1,ADR2)

C      specification: amener en memoire virtuelle la page
C      "fichier" "directory" qui suit la page "fichier"
C      "directory" implantee en memoire virtuelle a l'a-
C      dresse adr2 ( ou si celle-ci est la derniere de la
C      de la table tpb2,creer une nouvelle page "fichier"
C      "directory" et creer la page virtuelle "directory"
C      qui lui correspond) et creer la page "directory"
C      correspondante de la table tpb1 a la suite de la
C      derniere page [d'adresse adr1] de cette table tpb1.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PD1DIM,PGDIM,
     C NBPGMV,PRIMAX
      PRPG = PRIMAX + 1
      INT = 0
C      allocation de deux pages virtuelles.
      CALL PGMGET (FADR1)
```

```
      FAD1 = ADCA + (((FADR1 - ADMV) / PGDIM) * 2)
       CALL PGMSMP (FAD1,INT,PRPG,INT,INT,INT,INT)
       CALL PGMGET (FADR2)
      FAD2 = ADCA + (((FADR2 - ADMV) / PGDIM) * 2)
       CALL PGMSMP (FAD2,INT,PRPG,INT,INT,INT,INT)
       ADR = ADR2 + 1

C        decodage de l'entete de la page de trb2 d'adresse adr2.

       CALL WORDCD (ADR2,TF,BLK)
       CALL WOLEGE (ADR,FFP)
      IF (FFP.NE.0) 111,112

C        recherche de la page suivante de la table trb2.

  111  CALL PGFIRD (TF,FFP,FADR2)
       CALL WORCOD (FADR2,TF,BLK)
       GOTO 113

C        etant a la fin de la table trb2, on ajoute une
C        nouvelle page a cette table.
C        allocation d'une page "fichier".

  112  CALL PGFGET (TF,FFP)

C        mise a jour du pointeur de chainage entre les pages
C        "fichier" de la table trb2.

       CALL WOLEPU (ADR,FFP)

C        creation de l'entete de la nouvelle page de
C        la table trb2.

       CALL WORCOD (FADR2,TF,BLK)
       ADR = FADR2 + 1
       FP = FFP
       FFP = 0
       CALL WORCOD (ADR,FFP,FP)

C        creation de l'entete de la page correspondante
C        de la table trb1.

  113  ADR = ADR1 + 1
       CALL WORPUT (ADR,FADR1)
       CALL WORIPU (FADR1,FADR2)
       ADR = FADR1 + 1
       CALL WORPUT (ADR,INT)

C        sauvetage dans adr1 et adr2 des adresses des
C        nouvelles pages "directory" ainsi creees.

       ADR1 = FADR1
       ADR2 = FADR2
       RETURN
       END


      SUBROUTINE DIROUT (ADR1,ADR2)

C        specification:
C        recopier sur fichier la version virtuelle [si elle
C        existe] de la page "directory" qui suit la page
C        "directory"  implantee en memoire virtuelle a
C        l'adresse adr2 et desallouer les pages virtuelles
C        d'adresses adr1 et adr2.

       IMPLICIT INTEGER (A-U)
       COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     C  NBPGMV,PRIMAX
      COMMON /BMBM2/ ADRBUF,NUMBUF

C        decodage de l'entete de la page [d'adresse adr1] de
C        la table trb1.
```

```
          ADR = ADR1 + 1
          CALL WORGET (ADR,FADR1)
          IF (FADR1.NE.0) 121,122
  121     ADR = ADR2 + 1
          CALL WORISE (FADR1,FADR2)

C          decodage de l'entete de la page [d'adresse adr2] de
C          la table tpb2.

          CALL WOLEGE (ADR,FFP)
          CALL WORDCD (ADR2,TF,BLK)

C          recopie sur fichier de la page de la table tpb2.

         CALL PGCOPY (FADR2,ADRBUF)
          CALL PGFIWR (TF,FFP,ADRBUF)
          GOTO 123
  122     FADR2 = 0

C          liberation des pages virtuelles occupees par les
C          deux pages "directory".

  123     CALL PGMFRE (ADR1)
          CALL PGMFRE (ADR2)

C          sauvetage des adresses des pages suivantes des
C          tables tpb1 et tpb2.

          ADR1 = FADR1
          ADR2 = FADR2
          RETURN
          END


        FUNCTION LENGTH (STRING)

C          specification: calculer la longueur d'un string
C          qui est le nom d'un fichier et envoyer un message
C         d'erreur si ce nom a plus de 14 caracteres.

          IMPLICIT INTEGER (A-U)
         DIMENSION STRING (3)
         DO  550  J = 1 , 3
                  MOT = STRING(J)
                   DO  551  I = 1 , 5
                  CALL BMCHGE (N,MOT,I)
                  IF ((N.EQ.32).OR.(N.EQ.0)) GOTO 560
  551             CONTINUE
  550     CONTINUE
          TYPE 570 ,STRING
  570     FORMAT (' Le nom d''un fichier doit comprendre au plus',
     c    ' 14 caracteres',/,' et cette contrainte n''est pas ',
     c    'respectee pour le fichier dont le nom tronque a 14 ',
     c    'caracteres est [',2A5,A4,']')
          STOP
  560     LENGTH = ((J - 1) * 5) + I - 1
          RETURN
          END



        SUBROUTINE PGCOPY (ADA,ADB)

C          specification: copier le contenu de la page virtuelle
C          d'adresse ada dans la page virtuelle d'adresse adb.

          IMPLICIT INTEGER (A-U)
          COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c    NBPGMV,PRIMAX
         ADRA = ADA - 1
          ADRB = ADB - 1
          DO  520  I = 1 , PGDIM
                   ADRA = ADRA + 1
                   ADRB = ADRB + 1
```

```
                CALL WORGET (ADRA,INT)
                CALL WORPUT (ADRB,INT)
  520     CONTINUE
          RETURN
          END



        SUBROUTINE PGFFRE (TF,FP)

C          specification: desallouer la page no fp du fichier
C          identifie par le numero logique tf.

         IMPLICIT INTEGER (A-U)
          COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c    NBPGMV,PRIMAX
          FFP = 0

C          decodage de l'entete de la fenetre contenant
C          la derniere page de la table tplf d'un fichier.

          CALL WLLGET (ADTPLF,INT)
          IF (INT.NE.0) GOTO 501

C          la fenetre est vide: y est amenee la derniere page
C          de la table tplf du fichier no tf.

  510     CALL PGFIRD (TF,FFP,ADTPLF)

C          mise a jour de l'entete de la table tplf
C          avec le no du fichier.

          CALL WLLPUT (ADTPLF,TF)
          GOTO 502
  501     IF (INT.EQ.TF) GOTO 502

C          la fenetre n'est pas vide et elle contient la
C          derniere page de la table tplf du fichier no int.
C          ejection du contenu de cette page dans le fichier
C          no int et transfert ,dans la fenetre, de la dernie-
C          re page de la table tplf du fichier no tf.

          CALL PGFIWR (INT,FFP,ADTPLF)
          GOTO 510
  502     CALL WLRGET (ADTPLF,LENT)

C          le nombre de pages libres pouvant etre referencees
C          dans une page de la table tplf est donne par nb.

          NB = PGDIM - 2
          IF (LENT.EQ.NB) 503,504

C          la table tplf est remplie: il faut lui ajouter
C          une page et l'on procede comme suit:

C          le contenu de la fenetre est transfere dans la
C          page qui devait etre desallouee [no fp] et la
C          nouvelle page de la table tplf est initialisee
C          dans la fenetre.

  503     CALL WORGET (ADTPLF + 1,INT)
          CALL PGFIWR (TF,FP,ADTPLF)
          CALL WORPUT (ADTPLF + 1,INT)
           INT = 0
          CALL WLLPUT (ADTPLF,TF)
           CALL WLRPUT (ADTPLF,INT)
           CALL WORIPU (ADTPLF,FP)
          RETURN

C          la derniere page de la table tplf n'est pas remplie:
C          le numero de la page qu'on desalloue y est note a la
C          suite de la derniere entree active.

  504     ADR = ADTPLF + LENT + 2
          CALL WORPUT (ADR,FP)
```

```
      LENT = LENT + 1
      CALL WLRPUT (ADTPLF,LENT)
      RETURN
      END


      SUBROUTINE PGFGET (TF,FP)

C          specification: allouer une page appartenant au fichier
C          dont le numero d'identification logique est tf.

      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c NBPGMV,PRIMAX

C          decodage de l'entete de la fenetre destinee a recevoir
C          la derniere page de la table tplf d'un fichier.

      CALL WLLGET (ADTPLF,INT)
      IF (INT.NE.0) GOTO 401

C          la fenetre est vide: y est amenee la derniere page de
C          la table tplf du fichier no tf.

      FP = 0
      CALL PGFIRD (TF,FP,ADTPLF)

C          l'entete est mise a jour avec le numero
C          logique du fichier.

      CALL WLLPUT (ADTPLF,TF)
      GOTO 402

C          la fenetre n'est pas vide.

  401 IF (INT.EQ.TF) GOTO 402

C          la fenetre ne reference pas la table tplf
C          du fichier no tf.

      FP = 0

C          ejection de son contenu dans le fichier no int.

      CALL PGFIWR (INT,FP,ADTPLF)

C          transfert ,dans la fenetre, de la derniere page
C          de la table tplf du fichier no tf.

      CALL PGFIRD (TF,FP,ADTPLF)

C          mise a jour de l'entete de la table tplf avec le
C          numero du fichier [tf].

      CALL WLLPUT (ADTPLF,TF)
  402 CALL WLRGET (ADTPLF,LENT)

C          lent represente le nombre d'entrees actives dans la
C          derniere page de la table tplf.

      IF (LENT.EQ.0) 403,404

C          la fenetre ne reference aucune page libre.

  403 CALL WORIGE (ADTPLF,PT)

C          pt est un pointeur vers la page precedente de la table
C          tplf [il est nul si elle n'existe pas]

      IF (PT.EQ.0) 405,406

C          le fichier est rempli.
C          creation d'une nouvelle page.
```

```
  405     ADR = ADTPLF + 1
          CALL WORIGE (ADR,NBFIPG)
          FP = NBFIPG
         NBFIPG = NBFIPG + 1
         CALL WORIPU (ADR,NBFIPG)
          RETURN

C          le fichier n'est pas rempli.
C          transfert ,dans la fenetre, de l'avant-derniere
C          page de la table tplf et allocation de la page
C          "fichier" qu'elle occupait.

  406     FP = PT
          CALL PGFIRD (TF,PT,ADTPLF)
          CALL WLLPUT (ADTPLF,TF)
          RETURN

C          la fenetre reference des pages libres: allocation
C          de la page renseignee dans la derniere entree active .

  404     ADR = ADTPLF + LENT + 1
          CALL WORGET (ADR,FP)
         LENT = LENT - 1
         CALL WLRPUT (ADTPLF,LENT)
          RETURN
          END



        SUBROUTINE PGFIRD (TF,FP,ADR)

C          specification: transferer la page no fp du fichier
C          identifie par le numero logique tf dans la page
C         virtuelle d'adresse adr.

         IMPLICIT INTEGER (A-U)
         COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     C    NBPGMV,PRIMAX
         COMMON /BMBM2/ADRBUF,NUMBUF
          DIMENSION FINAME (3)

C          calcul de l'adresse de l'entree de la table
C          des fichiers qui reference le fichier no tf.

          AD = ADTF + ((TF - 1) * 4) + 3

C          recherche du numero d'identification
C          interne de ce fichier.

         CALL WORIGE (AD,JFN)

C          transfert de la page "fichier" no fp dans le
C         buffer d'adresse adrbuf.

          CALL BMPGRD (JFN,FP,NUMBUF,ERCODE)
          IF (ERCODE.EQ.0) GOTO 630

C          il s'est passe quelque chose d'anormal durant le
C          le transfert:  envoi d'un message d'erreur.

          NB = 6
          AD = AD - 4
          DO  629  I = 1 , 3
                   AD = AD + 1
                   CALL WORGET (AD,FINAME(I))
  629     CONTINUE
          CALL BMERFI (NB,FINAME,ERCODE)
          RETURN

C          copie du buffer dans la page d'adresse adr.

  630     CALL PGCOPY (ADRBUF,ADR)
          CALL BMUNMP (NUMBUF)
          RETURN
          END
```

```
      SUBROUTINE PGFIWR (TF,FP,ADR)

C          specification: transferer la page virtuelle d'adresse
C          adr dans la page no fp du fichier no tf.

          IMPLICIT INTEGER (A-U)
          DIMENSION FINAME (3)
           COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX
          COMMON /BMBM2/ADRBUF,NUMBUF

C          calcul de l'adresse de l'entree de la table
C          des fichiers correspondant au fichier no tf.

          AD = ADTF + ((TF - 1) * 4) + 3

C          recherche du numero d'identification
C          interne du fichier no tf.

          CALL WORIGE (AD,JFN)

C          transfert sur fichier de la page d'adresse adr.

          NP = ADR / PGDIM
          CALL BMPGWR (JFN,FP,NP,ERCODE)
           IF (ERCODE.EQ.0) GOTO 640

C          quelque chose d'anormal s'est passe durant le
C          transfert: ennvoi d'un message d'erreur.

          NB = 7
           AD = AD - 4
           DO  639  I = 1 , 3
                    AD = AD + 1
                    CALL WORGET (AD,FINAME(I))
  639      CONTINUE
           CALL BMERFI (NB,FINAME,ERCODE)
  640      RETURN
          END



      SUBROUTINE PGMFRE (ADR)

C          specification: desallouer la page virtuelle
C          d'adresse adr.

          IMPLICIT INTEGER (A-U)
          COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c     NBPGMV,PRIMAX

C          la page a desallouer est la page consignee dans
C          l'entree no j de la carte [d'adresse adc ].

          J = ((ADR - ADMV) / PGDIM) + 1
          ADC = ADCA + ((J - 1) * 2)
          INT = 0
          CALL WORPUT (ADC,INT)
          ADC = ADC + 1
          CALL WORPUT (ADC,INT)
          RETURN
          END



      SUBROUTINE PGMGET (ADR)

C          specification: allouer une page virtuelle et mettre
C          son adresse dans la variable adr.
```

```
      IMPLICIT INTEGER (A-U)
      COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     c NBPGMV,PRIMAX
      INT = 0
     J = 0
     K = 0

C       recherche de la priorite de la premiere page virtuelle
C       renseignee dans la carte.

      ADR = ADCA - 2
  168  J = J + 1
      ADR = ADR + 2
     K = K + 1
     CALL BYPPGE (ADR,PRPG)

C       si cette priorite est nulle, la page est allouee.

     IF (PRPG.EQ.0) 161,162

C       si, a l'issue du parcours sequentiel de la carte,
C       aucune page de priorite nulle n'a ete trouvee,
C       la page selectionnee est celle dont les priorites
C       de page et de bloc sont minimales [la cle primaire
C       etant prpg (int1) et la cle secondaire prblk (int2) ].

  162  IF (PRPG.EQ.(PRIMAX + 1)) GOTO 168
      CALL BYPBGE (ADR,PRBLK)
      INT1 = PRPG
      INT2 = PRBLK

C       la variable adc represente l'adresse de l'entree
C       de la carte qui reference la page selectionnee.

      ADC = ADR
     K = K + 1
     DO  163   I = K , NBPGMV
               ADR = ADCA + (I - 1)*2
               CALL BYPPGE (ADR,PRPG)
               IF (PRPG.EQ.0) 164,165
  164           J = I
               GOTO 161
  165           IF ((PRPG - INT1)) 166,167,163
  166           CALL BYPBGE (ADR,PRBLK)
               INT1 = PRPG
 1660          INT2 = PRBLK
               ADC = ADR
               J = I
               GOTO 163
  167          CALL BYPBGE (ADR,PRBLK)
               IF (PRBLK.LT.INT2) 1660,163
  163  CONTINUE
     CALL BYPTGE (ADC,PT1)
      CALL WORGET (PT1,ADR)
     CALL WORPUT (PT1,INT)
     CALL BYFLGE (ADC,FLAG)
     IF (FLAG.NE.0) 169,161

C       la page selectionnee est occupee par une page
C       d'un autre bloc et comme elle a ete modifiee
C       [flag = 1] ,elle doit faire l'objet d'une
C       recopie sur fichier.

  169  CALL BYFIGE (ADC,TF,FP)
      CALL PGFIWR (TF,FP,ADR)

C       calcul de l'adresse de la page selectionnee.

  161  ADR = ((J - 1) * PGDIM) + ADMV
      DO  170   I = 1 , PGDIM
               AD = ADR + I - 1
               CALL WORPUT (AD,INT)
  170  CONTINUE
     CALL DATMAP (ADR)
     RETURN
      END
```

```
      SUBROUTINE BMERBL (NB,BLKID)

C          specification: envoyer un message d'erreur concernant
C          la manipulation incorrecte d'un bloc dans le traite-
C        correspondant au numero nb et arreter le programme
C        en cours.

         IMPLICIT INTEGER (A-U)
        DIMENSION FINAME (3)
         COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     cNBPGMV,PRIMAX
         IF (NB.GT.1) GOTO 599
         CALL WIDDCD (BLKID,BLID,NIBLK,FLAG)
         CALL WORIGE (BLID,ADR2)
         CALL WORDCD (ADR2,TF,BLK)
         GOTO 600
  599    CALL WIDDCD (BLKID,TF,BLK,FLAG)
  600    ADR = ADTF + ((TF - 1) * 4) - 1
        DO  601  I = 1 , 3
                 ADR = ADR + 1
                 CALL WORGET (ADR,FINAME (I))
  601    CONTINUE
        CALL TRACE
        GOTO (1000,2000,3000,4000,7000,8000,9000),NB
 1000    PRINT 700,BLK,FINAME
        STOP
 2000    PRINT 701,BLK,FINAME
        STOP
 3000    PRINT 702,BLK,FINAME
         STOP
 4000    PRINT 703,BLK,FINAME
        STOP
 7000    PRINT 704,BLK,FINAME
         STOP
 8000    PRINT 705,BLK,FINAME
         STOP
 9000    PRINT 750,BLK,FINAME
         STOP

  700    FORMAT (' Demande illicite d''activation du bloc no',
     c    ' ',I3,' du fichier [',2A5,A4,']',/,' car ce bloc est',
     c    ' deja active.')

  701    FORMAT (' Le bloc no ',I3,' du fichier [',2A5,A4,']',
     c    ' est passe',/' comme parametre de la routine BLKCOP',
     c    ' et il n''est pas active',/,' alors qu''il devrait',
     c    ' l''etre.')

  702    FORMAT (' Le bloc no ',I3,' du fichier [',2A5,A4,']',
     c    ' est passe',/,' comme parametre de la routine BLKCUT',
     c    ' et il n''est pas active',/,' alors qu''il devrait',
     c    ' l''etre.')

  703    FORMAT (' Demande illicite de desactivation du bloc no',
     c    ' ',I3,' du fichier',/,' [',2A5,A4,']',
     c    ' car ce bloc est deja desactive.')

  704    FORMAT (' Le bloc no ',I3,' du fichier [',2A5,A4,'] ',
     c    'est passe',/,' comme parametre de la routine ITEMRD ',
     c    'et il n''est pas active',/,' alors qu''il devrait ',
     c    'l''etre.')

  705    FORMAT (' Le bloc no ',I3,' du fichier [',2A5,A4,']',
     c    ' est passe',/,' comme parametre de la routine ITEMWR',
     c    ' et il n''est pas active',/,' alors qu''il devrait ',
     c    'l''etre.')

  750    FORMAT (' La priorite accordee au bloc no ',I3,' du ',
     c    'fichier [',2A5,A4,']',/,' ne peut etre superieure',
     c    ' a la priorite de ses pages.')

         END
```

```
      SUBROUTINE BMERIT (NB,BLKID)

C          specification: envoyer un message d'erreur dans le
C          cas ou l'utilisateur introduit une dimension d'item
C          ou un numero d'item incorrects.

          IMPLICIT INTEGER (A-U)
          COMMON /BMBM1/ADCA,ADMV,ADTF,ADTPLF,NBFIUT,PDIDIM,PGDIM,
     cNBPGMV,PRIMAX
          DIMENSION FINAME (3)
         CALL WIDDCD (BLKID,BLID,NIBLK,FLAG)
         CALL WORIGE (BLID,ADR2)
         CALL WORDCD (ADR2,TF,BLK)
          ADR = ADTF + ((TF - 1) * 4) - 1
          DO   610   I = 1 , 3
                    ADR = ADR + 1
                     CALL WORGET (ADR,FINAME (I))
  610      CONTINUE
         CALL TRACE
          GOTO (1001,7001,7002,7005),NB
          ADR = ADR2 + 2
         CALL WORGET (ADR,ITDIM)
  1001     PRINT 706,BLK,FINAME,ITDIM
         STOP
  7001     IF (NIBLK.EQ.0) GOTO 7003
          PRINT 707,BLK,FINAME,NIBLK
          STOP
  7002     IF (NIBLK.EQ.0) GOTO 7004
          PRINT 708,BLK,FINAME,NIBLK,NIBLK
          STOP
  7005     PRINT 740,BLK,FINAME
         STOP
  7003     PRINT 731,BLK,FINAME
         STOP
  7004     PRINT 732,BLK,FINAME
         STOP
   706     FORMAT (' Lors de l''activation du bloc no ',I3,' du ',
     c     'fichier [',2A5,A4,']:',/,' la dimension communiquee',
     c     I3,' des items de ce bloc s''est revelee incorrecte.',/,
     c     ' La dimension (en mots) exacte est la suivante:[',
     c     I5,'].')

   707     FORMAT (' Demande incorrecte de lecture d''un item',
     c     ' du bloc no ',I3,' du fichier [',2A5,A4,']',/,' le',
     c     ' numero de l''item en question n''existe pas. Pour',
     c     ' etre valable,',/,' il doit etre compris entre [1',
     c     ' et ',I8,'].')

   708     FORMAT (' Demande incorrecte d''ecriture d''un item',
     c     ' du bloc no ',I3,' du fichier [',2A5,A4,']',/,
     c     ' le numero de l''item en question n''est pas va',
     c     'lable.',/,' Si vous desirez modifier un item, vous ',
     c     'devez communiquer un numero compris entre [ 1 et ',
     c     I8,/,' et si vous desirez creer un nouvel item, son ',
     c     'numero doit etre le nombre',/,' d''items du bloc',
     c     ' (',I8,') augmente de 1.')

   731     FORMAT (' Le bloc no ',I3,' du fichier [',2A5,A4,
     c     '] ne contient aucun item',/,' et l''appel au modu',
     c     'le ITEMRD est par consequent illicite.")

   732     FORMAT (' Vous desirez creer un item dans le bloc no ',
     c     I3,/,' du fichier [',2A5,A4,'] et dans le cas present',
     c     ', ce bloc est vide',/,' et le numero du premier item',
     c     ' cree doit donc etre 1.')

   740     FORMAT (' Vous desirez tronquer le bloc no ',I3,' du',
     c     ' fichier [',2A5,A4,']',/' a partir d''un item qui ',
     c     'n''existe pas.')
          END
```

```
      SUBROUTINE BMERFI (NB,FINAME,ERCODE)

C          specification: envoyer un message d'erreur lors de l'echec
C          d'une operation sur un fichier et arreter le programme en cours.

         IMPLICIT INTEGER (A-U)
         DIMENSION FINAME (3)
         CALL TRACE
          GOTO (8001,8002,8003,8004,8005,8006,8007),NB
 8001     PRINT 709,FINAME
         STOP
 8002     PRINT 710,FINAME
         STOP
 8003     PRINT 711,FINAME
         STOP
 8004     GOTO (1006,1007,1008),ERCODE
 8005     PRINT 712,FINAME,ERCODE
         STOP
 8006     PRINT 728,FINAME
         STOP
 8007     PRINT 729,FINAME
         STOP
 1006     PRINT 713,FINAME
          STOP
 1007     PRINT 714,FINAME
          STOP
 1008     PRINT 715,FINAME
          STOP
  709     FORMAT (' L''ouverture du fichier [',2A5,A4,'] n''a',
     c    ' pas pu s''effectuer correctement.')

  710     FORMAT (' La fermeture du fichier [',2A5,A4,'] n''a',
     c    ' pas pu s''effectuer correctement.')

  711     FORMAT (' La transformation du nom du fichier [',
     c    2A5,A4,'] en code ASCIZ a echoue.')

  712     FORMAT (' Le nombre de blocs du fichier [',2A5,A4,
     c    '] n''est pas celui communique:',/,'  ce fichier ',
     c    'contient en fait [',I4,'] blocs.')

  713     FORMAT (' Le fichier [',2A5,A4,'] existe deja ,',
     c    'alors qu''il a ete declare comme n''existant pas',
     c    ' encore.',/,'  Il faut donc le declarer avec la routi',
     c    'ne FILDEF et non avec la routine FILCRE.')
  714     FORMAT (' Le fichier [',2A5,A4,'] n''a pas ete trouve,',
     c    ' alors qu''il a ete declare comme existant deja.')

  715     FORMAT (' La recherche du JFN du fichier [',2A5,A4,
     c    '] a echoue.')

  728     FORMAT (' Une erreur s''est produite lors d''une exe',
     c    'cution de la routine BMPGRD',/,
     c    ' pour le fichier [',2A5,A4,']')

  729     FORMAT (' Une erreur s''est produite lors d''une exe',
     c    'cution de la routine BMPGWR',/,
     c    ' pour le fichier [',2A5,A4,']')
         END



      SUBROUTINE BMERME (NB,ERCODE)

C          specification: emettre un message indiquant l'impossibilite
C          d'obtenir la place memoire demandee.

         IMPLICIT INTEGER (A-U)

         CALL TRACE
          GOTO (9001,9002,9003),ERCODE
 9001     GOTO (90041,90051,90061,90071),NB
 9002     GOTO (90042,90052,90062,90072),NB
 9003     GOTO (90043,90053,90063,90073),NB
```

```
90041     PRINT 716
          STOP
90051     PRINT 717
          STOP
90061     PRINT 718
          STOP
90071     PRINT 719
          STOP
90042     PRINT 720
          STOP
90052     PRINT 721
          STOP
90062     PRINT 722
          STOP
90072     PRINT 723
          STOP
90043     PRINT 724
          STOP
90053     PRINT 725
          STOP
90063     PRINT 726
          STOP
90073     PRINT 727
          STOP

  716     FORMAT (' Echec dans l''execution de la routine ALLOCA.',/,
     c' il n''y pas assez de memoire disponible dans le systeme.')

  717     FORMAT (' Echec dans l''execution de la routine MAPCRE.',/,
     c' il n''y pas assez de memoire disponible dans le systeme.')

  718     FORMAT (' Echec dans l''execution de la routine TFICRE.',/
     c' il n''y a pas assez de memoire disponible dans le systeme.')

  719     FORMAT (' Echec dans l''execution de la routine TPLFCR.',/
     c' il n''y pas assez de memoire disponible dans le systeme.')

  720     FORMAT (' Echec dans l''execution de la routine ALLOCA.',/
     c' la reservation de la place memoire ne peut se faire a partir'
     c,' de l''adresse demandee.')

  721     FORMAT (' Echec dans l''execution de la routine MAPCRE.',/
     c' la reservation de la place memoire ne peut se faire a partir'
     c,' de l''adresse demandee.')

  722     FORMAT (' Echec dans l''execution de la routine TFICRE.',/
     c' la reservation de la place memoire ne peut se faire a partir'
     c,' de l''adresse demandee.')

  723     FORMAT (' Echec dans l''execution de la routine TPLFCR.',/
     c' la reservation de la place memoire ne peut se faire a partir'
     c,' de l''adresse demandee.')

  724     FORMAT (' Echec dans l''execution de la routine ALLOCA.',/
     c' les arguments communiques sont illegaux.')

  725     FORMAT (' Echec dans l''execution de la routine MAPCRE.',/
     c' les arguments communiques sont illegaux.')

  726     FORMAT (' Echec dans l''execution de la routine TFICRE.',/
     c' les arguments communiques sont illegaux.')

  727     FORMAT (' Echec dans l''execution de la routine TPLFCR.',/
     c' les arguments communiques sont illegaux.')
          END


      TITLE BYFIGE

;SUBROUTINE BYFIGE (adr,tf,fp) - mettre dans tf et fp le
;numero de fichier et le numero de la page "fichier" cor-
;respondant a la page virtuelle renseignee dans l'entree
;Ed'adresse adr] de la carte.
;une entree de la carte a la structure suivante:
;flag   /  prpg  /  prblk   /   tf   /    fp   /  pti
```

```
;0      1      9  10      17  18    35 0     17  18    35
;arguments donnes par l'accumulateur no 16.
;mot 0: adr
;mot 1: tf
;mot 2: fp

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BYFIGE
BYFISE:MOVE     1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse de l'entree de la carte.
        MOVE    2,1
         HRRZ   1,0(1)    ;mettre dans le registre no 1
                          ;le numero du fichier.
        MOVEM   1,@1(16)
         HLRZ   2,1(2)    ;mettre dans le registre no 2
                          ;le numero de la page "fichier".
        MOVEM   2,@2(16)
        POPJ    17,0
        END

        TITLE BYFLGE

;SUBROUTINE BYFLGE (adr,flag) -  mettre dans flag la valeur de
;l'indicateur de modification de page correspondant a l'entree
;de la carte d'adresse adr.
;une entree de la carte a la structure suivante:
;flag   /   prpg   /  prblk   /  tf    /  fp   /   pt1
;0      1        9  10      17 18     35 0    17 18    35
;arguments donnes par l'accumulateur no 16.
;mot 0: adr
;mot 1: flag.

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BYFLGE
BYFLGE:MOVE     1,@0(16)   ;mettre dans le registre no 1
                           ;l'adresse de l'entree de la carte.
        LDB     2,[POINT 1,0(1),0]
                           ;mettre dans le registre no 2
                           ;le bit no 0 du mot pointe par adr.
        MOVEM   2,@1(16)
        POPJ    17,0
        END

        TITLE BYFLPU

;SUBROUTINE BYFLPU (adr,flag) - modifier la valeur de l'indi-
;cateur de modification de page correspondant a l'entree de
;la carte d'adresse adr.
;arguments donnes par l'accumulateur no 16.
;mot 0: adr
;mot 1: flag

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BYFLPU
BYFLPU:MOVE     1,@0(16)   ;mettre dans le registre no 1
                           ;l'adresse de l'entree de la carte.
        MOVE    2,@1(16)   ;mettre dans le registre no 2
                           ;le contenu de flag [=1].
        DPB     2,[POINT 1,0(1),0]
                           ;mettre a 1 le bit no 0 du mot
                           ;pointe par adr.
        POPJ    17,0
        END

        TITLE BYPRGE

;SUBROUTINE (adr,prblk) - mettre dans prblk la priorite du bloc
```

```
;contenant la page referencee dans l'entree [d'adresse adr] de
;la carte.
;une entree de la carte a la structure suivante:
;  flag  /  prpg  /  prblk  /   tf  /  fp   /  pt1
;0       1       9  10     17 18    35 0    17 18   35
;arguments donnes par le registre no 16.
;mot 0 : adr
;mot 1 : prblk

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BYPBGE
BYPBGE:MOVE     1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse de l'entree de la carte.
        LDB     2,[POINT 8,0(1),17]
                          ;mettre dans le registre no 2 les
                          ;bits 10 a 17 du mot pointe par adr.
        MOVEM   2,@1(16)
        POPJ    17,0
        END


        TITLE BYPPGE

;SUBROUTINE BYPPGE (adr,prpg) - mettre dans prpg la priorite de
;la page virtuelle renseignee dans l'entree de la carte, dont
;l'adresse est adr.
;une entree de la carte a la structure suivante:
;flag  /  prpg  /prblk  /  tf   /  fp  /  pt1
;0     1       9 10    17 18    35 0   17 18   35
;arguments donnes par l'accumulateur par no 16.
;mot 0: adr
;mot 1: prpg

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BYPPGE
BYPPGE:MOVE     1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse de l'entree de la carte.
        LDB     2,[POINT 9,0(1),9]
                          ;mettre dans le registre no 2 les
                          ;bits 1 a 9 du mot pointe par adr.
        MOVEM   2,@1(16)
        POPJ    17,0
        END


        TITLE BYPPPU

;SUBROUTINE BYPPPU (adr,prpg) - assigner la priorite prpg a la
;page virtuelle consignee dans l'entree [d'adresse adr] de la
;carte.
;une entree de la carte a la structure suivante:
;  flag  /  prpg  /  prblk  / tf  /  fp  /  pt1
;0       1       9 10      17 18   35 0   17 18   35
;arguments donnes par le registre no 16.
;mot 0: adr
;mot 1: prpg

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BYPPPU
BYPPPU:MOVE     1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse de l'entree de la carte.
        MOVE    2,@1(16)  ;mettre dans le registre no 2 la
                          ;valeur de la priorite.
        DPB     2,[POINT 9,0(1),9]
                          ;sauver la priorite dans les bits
                          ;1 a 9 du mot pointe par adr.
        POPJ    17,0
        END
```

```
        TITLE BYPTGE

;SUBROUTINE BYPTGE (adr,pt1) - mettre dans pt1 le pointeur
;vers l'entree de la table tpbi correspondant a la page vir-
;tuelle renseignee dans l'entree, d'adresse adr de la carte.
;une entree de la carte a la structure suivante:
;flag  /prpg   /  prblk  /  tf   /   fp   /  pti
;0     1     9  10     17  18    35 0   17   18   35
;arguments donnes par l'accumulateur no 16.
;mot 0: adr
;mot 1: pt1

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BYPTGE
BYPTGE: MOVE    1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse de l'entree de la carte.
        HRRZ    1,1(1)    ;mettre dans le registre no 1
                          ;la valeur du pointeur pt1.
        MOVEM   1,@1(16)
        POPJ    17,0
        END


        TITLE BMCHGE
;SUBROUTINE BMCHGE (n,word,l) - mettre dans la variable n le
;caractere no l du mot word.
;arguments donnes par le registre no 16.
;mot 0 : n
;mot 1 : word
;mot 2 : l

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BMCHGE
BMCHGE: MOVE    2,@2(16)  ;mettre dans le registre no 2
                          ;le numero l.
        MOVE    1,[POINT 7,@1(16)]
                          ;mettre dans le registre no 1 le byte
                          ;pointer vers le mot word.
        MOVEI   3,0
COUNT:  ADDI    3,1
        ILDB    4,1       ;mettre dans le registre no 4 le
                          ;caractere dont le numero est le
                          ;contenu du registre no 3.
        CAME    3,2       ;comparer le contenu du registre no 3
                          ;avec l'entier l .
        JUMPA   COUNT
        MOVEI   3,0
        DPB     4,[POINT 7,@0(16),35]
                          ;garnir les 7 derniers bits du mot n
                          ;avec le code asci du caractere no l
                          ;du mot word.
        DPB     3,[POINT 29,@0(16),28]
                          ;mettre a zero les autres bits du mot n.
        POPJ    17,0
        END


        TITLE PGMSMP

;SUBROUTINE PGMSMP (adr,flag,prpg,prblk,tf,fp,pt1)
;mettre a jour l'entree [d'adresse adr] de la carte.
;arguments donnes par l'accumulateur no 16.
;mot 0: adr
;mot 1: flag.
;mot 2: prpg.
;mot 3: prblk.
;mot 4: tf.
;mot 5: fp.
;mot 6: pt1.

        SEARCH MACSYM,MONSYM
        SALL
```

```
        ENTRY   PGMSMP
PGMSMP: MOVE    1,@0(16)    ;mettre dans le registre no 1
                            ;l'adresse de l'entrre de la carte.
        MOVE    2,@1(16)    ;mettre le contenu de flag dans
                            ;le registre no 2.
        DPB     2,[POINT 1,0(1),0]
                            ;mettre le contenu du registre no 2
                            ;dans le bit no 0 de l'entree de
                            ;la carte.
        MOVE    2,@2(16)    ;mettre la priorite de le page
                            ;dans le registre no 2.
        DPB     2,[POINT 9,0(1),9]
                            ;mettre le contenu du registre no 2
                            ;dans les bits 1 a 9 de l'entree de
                            ;la carte.
        MOVE    2,@3(16)    ;mettre la priorite de bloc dans le
                            ;registre no 2.
        DPB     2,[POINT 8,0(1),17]
                            ;mettre le contenu du registre no 2
                            ;dans les bits 9 a 17 de l'entree de
                            ;la carte.
        MOVE    2,@4(16)    ;mettre le numero de fichier dans le
                            ;registre no 2.
        HRRM    2,0(1)      ;mettre le contenu du registre no 2
                            ;dans le demi-mot droit de l'entree
                            ;de la carte.
        MOVE    2,@5(16)    ;mettre dans le registre no 2
                            ;le numero de page "fichier".
        HRLM    2,1(1)      ;mettre le contenu du regisre no 2
                            ;dans le demi-mot gauche du second
                            ;mot de l'entree de la carte.
        MOVE    2,@6(16)    ;mettre dans le registre no 2
                            ;le pointeur pti.
        HRRM    2,1(1)      ;mettre le contenu du registre no 2
                            ;dans le demi-mot droit du second
                            ;mot de l'entree de la carte.
        POPJ    17,0
        END

        TITLE WIDCOD

;SUBROUTINE WIDCOD (blkid,blid,niblk,flag)
;disposer de la facon suivante les variables blid,niblk et flag
;dans le mot designe par blkid:
;blid    /   niblk    /  flag
;0     17  18       34     35
;arguments donnes par l'accumulateur no 16.
;mot 0: blkid
;mot 1: blid
;mot 2: niblk
;mot 3: flag

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WIDCOD
WIDCOD: MOVE    1,@1(16)    ;mettre dans le registre no 1
                            ;le contenu de blid.
        HRLZM   1,@0(16)    ;mettre dans la partie gauche de
                            ;blkid le contenu du registre no 1.
        MOVE    1,@2(16)    ;mettre dans le registre no 1
                            ;le contenu de niblk.
        DPB     1,[POINT 17,@0(16),34]
                            ;deposer le contenu du registre no 1
                            ;dans les bits 18 a 34 de blkid.
        MOVE    1,@3(16)    ;mettre dans le registre no 1
                            ;le contenu de flag.
        DPB     1,[POINT 1,@0(16),35]
                            ;deposer le contenu du registre no 1
                            ;dans le bit no 35 de blkid.
        POPJ    17,0
        END
```

```
        TITLE WIDDCD

;SUBROUTINE WIDDCD (blkid,blid,niblk,flag)
;decoder la variable blkid et mettre dans blid le 1/2 mot
;gauche, dans niblk les 17 bits suivants et dans flag le
;dernier bit.
;arguments donnes par l'accumulateur no 16.
;mot 0: blkid
;mot 1: blid
;mot 2: niblk
;mot 3: flag

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WIDDCD
 WIDDCD:HLRZ    1,@0(16)    ;mettre dans le registre no 1
                            ;la partie gauche de blkid.
        MOVEM   1,@1(16)
        LDB     1,[POINT 17,@0(16),34]
                            ;charger dans le registre no 1
                            ;le byte correspondant a niblk.
        MOVEM   1,@2(16)
        LDB     1,[POINT 1,@0(16),35]
                            ;charger dans le registre no 1
                            ;le byte correspondant a flag.
        MOVEM   1,@3(16)
        POPJ    17,0
        END


        TITLE WLLGET

;SUBROUTINE WLLGET (adr,var)
;mettre dans la variable var le contenu
;des huit premiers bits du mot pointe par adr.
;arguments donnes par le registre no 16.
;mot 0 : adr
;mot 1 : var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WLLGET
 WLLGET:MOVE    1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse du mot pointe par adr.
        LDB     2,[POINT 9,0(1),8]
                          ;charger dans le registre no 2
                          ;les bits 0 a 8 du mot pointe par adr.
        MOVEM   2,@1(16)
        POPJ    17,0
        END


        TITLE WLLPUT

;SUBROUTINE WLLPUT (adr,var)
;mettre le contenu de la variable var dans
;les huit premiers bits du mot pointe par adr.
;arguments donnes par le registre no 16.
;mot 0 : adr
;mot 1 : var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WLLPUT
 WLLPUT:MOVE    1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse du mot pointe par adr.
        MOVE    2,@1(16)  ;mettre le contenu de la variable
                          ;var dans le registre no 2.
        DPB     2,[POINT 9,0(1),8]
                          ;deposer le contenu du registre no 2
                          ;dans les bits 0 a 8 du mot pointe
                          ;par le mot adr.
        POPJ    17,0
```

```
        END

        TITLE WLRGET

;SUBROUTINE (adr,var)
;mettre dans la variable var le contenu des bits
;9 a 17 du mot pointe par adr.
;arguments donnes par le registre no 16.
;mot 0 : adr
;mot 1 : var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WLRGET
 WLRGET:MOVE    1,@0(16)  ;mettre l'adresse du mot pointe par
                          ;adr dans le registre no 1.
        LDB     2,[POINT 9,0(1),17]
                          ;charger dans le registre no 2 les
                          ;9 a 17 du mot pointe par adr.
        MOVEM   2,@1(16)
        POPJ    17,0
        END

        TITLE WLRPUT

;SUBROUTINE WLRPUT (adr,var)
;mettre le contenu de la variable var dans
;les bits 9 a 17 du mot pointe par adr.
;arguments donnes par le registre no 16.
;mot 0 : adr
;mot 1 : var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WLRPUT
 WLRPUT:MOVE    1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse du mot pointe par adr.
        MOVE    2,@1(16)  ;mettre dans le registre no 2
                          ;le contenu de la variable var.
        DPB     2,[POINT 9,0(1),17]
                          ;deposer le contenu du registre
                          ;no 2 dans les bits 9 a 17 du mot
                          ;pointe par adr.
        POPJ    17,0
        END

        TITLE WOLEGE

;SUBROUTINE WOLEGE (adr,var)
;mettre le demi-mot gauche du mot pointe
;par adr dans la variable var.
;arguments donnes par le registre no 16.
;mot 0 : adr
;mot 1 : var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WOLEGE
 WOLEGE:MOVE    1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse du mot pointe par adr.
        MOVE    1,0(1)    ;mettre dans le registre no 1
                          ;le contenu du mot pointe par adr.
        HLRZM   1,@1(16)
        POPJ    17,0
        END

        TITLE WOLEPU

;SUBROUTINE WOLEPU (adr,var)
```

```
;mettre le contenu de la variable var dans
;le demi-mot gauche du mot pointe par adr.
;arguments donnes par le registre no 16.
;mot 0 : adr
;mot 1 : var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WOLEPU
WOLEPU: MOVE    1,@0(16)    ;mettre dans le registre no 1
                            ;l'adresse du mot pointe par adr.
        HRRZ    2,@1(16)    ;mettre dans le registre no 2
                            ;le contenu de la variable var.
        HRLM    2,0(1)      ;mettre le contenu du registre no 2
                            ;dans la partie gauche du mot pointe
                            ;par adr.
        POPJ    17,0
        END


        TITLE WORCOD

;SUBROUTINE WORCOD (adr,m1,m2)
;mettre dans le demi-mot gauche [droit]
;du mot pointe par le mot adr le mot m1 [m2].
;arguments donnes par le registre cx.
;mot 0 : adr
;mot 1 : m1
;mot 2 : m2

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WORCOD
WORCOD: MOVE    1,@1(16)    ;mettre dans le registre no 1
                            ;le contenu du mot m1.
        MOVE    2,@0(16)    ;mettre dans le registre no 2
                            ;l'adresse du mot pointe par adr.
        HRLM    1,0(2)      ;mettre le contenu du registre no 1
                            ;dans la partie gauche du mot
                            ;pointe par adr.
        MOVE    1,@2(16)    ;mettre dans le registre no 1 le
                            ;contenu du mot m2.
        HRRM    1,0(2)      ;mettre le contenu du registre no 1
                            ;dans la partie droite du mot
                            ;pointe par adr.
        POPJ    17,0
        END


        TITLE WORDCD

;SUBROUTINE WORDCD (adr,m1,m2)
;mettre dans le mot m1 [m2] le contenu du demi-mot
;gauche [droit] du mot pointe par le mot adr.
;arguments donnes par le registre no 16.
;mot 0 : adr
;mot 1 : m1
;mot 2 : m2

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WORDCD
WORDCD: MOVE    1,@0(16)    ;mettre dans le registre no 1
                            ;l'adresse du mot pointe
                            ;par adr.
        HLRZ    3,0(1)      ;mettre dans le registre no 3
                            ;la partie gauche du mot pointe
                            ;par adr.
        HRRZ    2,0(1)      ;mettre dans le registre no 2
                            ;la partie droite du mot pointe
                            ;par adr.
        MOVEM   3,@1(16)
        MOVEM   2,@2(16)
```

```
        POPJ    17,0
        END

        TITLE WORGET

;SUBROUTINE WORGET (adr,var)
;mettre dans la variable var le contenu
;du mot pointe par le mot adr.
;arguments donnes par l'accumulateur no 16.
;mot 0: adr
;mot 1: var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WORGET
WORGET:MOVE     1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse du mot pointe par adr.
        MOVE    1,0(1)    ;mettre dans le registre no 1
                          ;le contenu du mot pointe par adr.
        MOVEM   1,@1(16)
        POPJ    17,0
        END

        TITLE WORIGE

;SUBROUTINE WORIGE (adr,var)
;mettre dans la variable var le contenu du
;demi-mot droit du mot pointe par adr.
;arguments donnes par le registre no 16.
;mot 0 : adr
;mot 1 : var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WORIGE
WORIGE:MOVE     1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse du mot pointe par adr.
        MOVE    1,0(1)    ;mettre dans le registre no 1 le
                          ;contenu du mot pointe par adr.
        HRRZM   1,@1(16)  ;mettre la partie droite du
                          ;registre no 1 dans la variable var.
        POPJ    17,0
        END

        TITLE WORIPU

;SUBROUTINE WORIPU (adr,var)
;mettre le contenu de la variable var dans
;le demi-mot droit du mot pointe par adr.
;mot 0 : adr
;mot 1 : var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WORIPU
WORIPU:MOVE     1,@0(16)  ;mettre dans le registre no 1
                          ;l'adresse du mot pointe par adr.
        HRRZ    2,@1(16)  ;mettre le contenu de la variable
                          ;var dans le registre no 2.
        HRRM    2,0(1)    ;mettre le contenu du registre no 2
                          ;dans la partie droite du mot
                          ;pointe par adr.
        POPJ    17,0
        END

        TITLE WORPUT

;SUBROUTINE WORPUT (adr,var)
;mettre le contenu de la variable var dans
```

```
;le mot pointe par le mot adr.
;arguments donnes par le registre no 16.
;mot 0: adr
;mot 1: var

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   WORPUT
WORPUT: MOVE    1,@1(16)  ;mettre dans le registre no 1
                          ;le contenu de la variable var.
        MOVE    2,@0(16)  ;mettre dans le registre no 2
                          ;l'adresse du mot pointe par adr.
        MOVEM   1,0(2)    ;mettre le contenu du registre no 1
                          ;dans le mot pointe par adr.
        POPJ    17,0
        END


        TITLE BMASCI

;SUBROUTINE BMASCI (soustr,length,desstr,ercode)
;transformer un string asci de longueur length
;en un string asciz.
;arguments donnes par le registre no 16.
;mot 0 : soustr
;mot 1 : length
;mot 2 : desstr
;mot 3 : ercode

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BMASCI
BMASCI: HRROI   1,@2(16)  ;mettre dans le registre no 1
                          ;le byte pointer vers le string
                          ;transforme en code asciz.
        HRROI   2,@0(16)  ;mettre dans le registre no 2
                          ;le byte pointer vers le string source.
        MOVN    3,@1(16)
        SOUT              ;appel superviseur.
        ERJMP   ERASCI
        SETZ    4,
        MOVEM   4,@3(16)
        IDPB    4,1       ;ajouter le caractere nul
                          ;au string destination.
        AOS     @1(16)    ;ajouter 1 a la longueur de ce string.
        POPJ    17,0

ERASCI: MOVEI   4,1
        POPJ    17,0
        END


        TITLE BMCLOS

;SUBROUTINE BMCLOS (jfn,ercode)
;fermer le fichier dont le job file number est jfn.
;arguments donnes par le registre no 16.
;mot 0 : jfn
;mot 1 : ercode :
;        = 0 :si la fermeture s'est effectuee correctement.
;        = 1 :si une erreur s'est produite.

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BMCLOS
BMCLOS: MOVEI   2,0
        HRLI    1,(CO%NRJ)  ;ne pas detruire le job file number.
        HRR     1,@0(16)
        CLOSF               ;fermeture du fichier.
        ERJMP   ERCLOS
        MOVEM   2,@1(16)
        POPJ    17,0
```

```
ERCLOS:MOVEI   2,1
       MOVEM   2,@1(16)
       POPJ    17,0
       END


       TITLE BMGFFA

;SUBROUTINE BMGFFA (admv)
;trouver l'adresse de debut de la zone libre.
;arguments donnes par l'accumulateur no 16.
;mot 0: admv

       SEARCH MACSYM,MONSYM
       SALL

       ENTRY   BMGFFA
       EXTERN  .JBFF
BMGFFA:HRRZ    1,.JBFF
       MOVEM   1,@0(16)
       POPJ    17,0
       END



       TITLE BMGJFN

;SUBROUTINE BMGJFN (string,jfn,flag,ercode)
;trouver le job file number [jfn] du fichier
;identifie par le nom string, le parametre
;flag valant 0 ou 1 selon que le fichier
;existe deja ou non.
;arguments donnes par le registre no 16.
;mot 0 : string
;mot 1 : jfn
;mot 2 : flag
;mot 3 : ercode :
;       = 0: il n'y a eu aucune erreur.
;       = 1: l'utilisateur a declare que le fichier
;            n'existait pas alors qu'il existe deja.
;       = 2: l'utilisateur a declare que le fichier
;             existait deja alors qu'il n'existe pas.
;       = 3: un type d'erreur a ete releve [autre que les
;            types precedents].

       SEARCH MACSYM,MONSYM
       SALL

       ENTRY   BMGJFN
BMGJFN:MOVE    4,@2(16)
       JUMPG   4,NEWFIL
       HRLZI   1,(GJ%OLD+GJ%SHT)
       JUMPA   FOLLOW
NEWFIL:HRLZI   1,(GJ%NEW+GJ%SHT)
FOLLOW:HRR0I   2,@0(16)    ;le pointeur vers le nom du fichier est
                           ;mis dans le registre no 2.
       GTJFN
       ERJMP   PROCED
       SETZM   @3(16)      ;le code d'erreur est mis a 0:
                           ;la recherche du jfn du fichier
                            ;s'est bien passee.
       HRRZM   1,@1(16)    ;le jfn est sauve dans le mot no 2.
       POPJ    17,0

PROCED:HRRZ    1,1          ;le numero de l'erreur est mis
                           ;dans le registre no 1.
       CAIE    4,1         ;mode de declaration du fichier:
                           ;ancien ou nouveau ?
       JUMPA   OLDFIL
       CAIE    1,GJFX27    ;le fichier a ete declare nouveau
       JUMPA   ERROR
       MOVEI   2,1         ;et il existe deja.
       MOVEM   2,@3(16)    ;mise a 1 du code d'erreur.
       POPJ    17,0
```

```
OLDFIL:CAIE     1,GJFX24    ;le fichier existerait deja
       JUMPA    ERROR
       MOVEI    2,2         ;et pourtant il n'a pas ete trouve.
       MOVEM    2,@3(16)    ;mise a 2 du code d'erreur.
       POPJ     17,0

 ERROR:MOVEI    2,3          ;une erreur d'un autre type que les
                             ;deux precedentes a ete detectee.
       MOVEM    2,@3(16)    ;mise a 3 du code d'erreur.
       POPJ     17,0
       END


       TITLE BAGMEM

;SUBROUTINE BAGMEM (admv,dim,ercode)
;allouer une zone memoire a partir de l'adresse donnee
;par admv et d'une taille donnee par dim.
;arguments donnes par l'accumulateur no 16.
;mot 0: admv
;mot 1: dim
;mot 3: ercode :
;       = 0 :la zone a ete allouee.
;       = 1 :il n'y a pas assez de memoire disponible
;            dans le systeme.
;       = 2 :la zone ne peut etre allouee a partir de
;            l'adresse demandee.
;       = 3 :les arguments sont illegaux.
;utilisation de la routine FUNCT. des "FOROTS".

       SEARCH MACSYM,MONSYM
       SALL
       ENTRY    BAGMEM
       EXTERN   FUNCT.
BAGMEM:MOVE     1,@0(16)   ;mettre dans le registre no 1
                           ;l'adresse de debut de la zone.
       HRRZM    1,ARG1
       MOVE     1,@1(16)   ;mettre dans le registre no 1
                           ;la taille de la zone desiree.
       HRRZM    1,ARG2
       MOVEM    16,SAVE16  ;sauver le contenu du registre 16.
       MOVEI    16,ARGS
       PUSHJ    17,FUNCT.

       MOVE     16,SAVE16  ;restaurer le contenu du registre 16.
       HRRZ     1,STATUS   ;sauver le resultat de l'operation.
       HRRZM    1,@2(16)
       POPJ     17,0

SAVE16:BLOCK    1

       -5,,0                ;arguments de la routine FUNCT.
  ARGS:0,,CODE
       0,,ERC
       0,,STATUS
       0,,ARG1
       0,,ARG2

  CODE:0,,1
   ERC:BLOCK    1
STATUS:BLOCK    1
  ARG1:BLOCK    1
  ARG2:BLOCK    1
       END


       TITLE BAOPEN

;SUBROUTINE BAOPEN (jfn,ercode)
;ouvrir le fichier dont le job file number est donne par jfn.
;arguments donnes par le registre no 16.
;mot 0 : jfn
;mot 1 : ercode :
;       = 0 : l'ouverture s'est effectuee correctement.
;       = 1 : il s'est passe quelque chose d'anormal.
```

```
        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BMOPEN
BMOPEN: HRRZ    1,@0(16)
        HRLI    2,(4485)
        MOVEI   4,0
        HRRI    2,OFZWR+OFZRD ;ouverture en lecture et
                              ;en ecriture.
        OPENF
        ERJMP   EROPEN
        MOVEM   4,@1(16)
        POPJ    17,0

EROPEN: MOVEI   4,1
        MOVEM 4,@1(16)
        POPJ  17,0
        END


        TITLE BMPGRD
;SUBROUTINE BMPGRD (jfn,fp,np,ercode)
;transferer la page "fichier" no fp du fichier identifie
;par jfn sur la page virtuelle no np.
;arguments donnes par le registre no 16.
;mot 0 : jfn
;mot 1 : fp
;mot 2 : np
;mot 3 : ercode

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BMPGRD
BMPGRD: HRL     1,@0(16)    ;mettre le jfn du fichier dans
                            ;la partie gauche du registre no 1.
        HRR     1,@1(16)    ;mettre le numero de la page "fichier"
                            ;dans la partie droite du registre no 1.
        HRLI    2,.FHSLF    ;mettre dans la partie gauche du regis-
                            ;tre no 2 le nom du processus courant.
        HRR     2,@2(16)    ;mettre dans la partie droite du
                            ;registre no 2 le numero de la
                            ;page virtuelle.
        HRLZI   3,(PM%RD+PM%WR)
        PMAP
        ERJMP   ERPMPR
        MOVEI   3,0
        MOVEM   3,@3(16)
        POPJ    17,0

ERPMPR: MOVEI   3,1
        MOVEM   3,@3(16)
        POPJ    17,0
        END


        TITLE BMPGWR
;SUBROUTINE BMPGWR (jfn,fp,np,ercode)
;transferer sur la page "fichier" no fp du fichier identifie
;par jfn la page virtuelle no np.
;arguments donnes par le registre no 16.
;mot 0 : jfn
;mot 1 : fp
;mot 2 : np
;mot 3 :ercode

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BMPGWR
BMPGWR: HRLI    1,.FHSLF    ;mettre dans la partie gauche du
                            ;registre no 1 le nom du processus
                            ;courant.
        HRR     1,@2(16)    ;mettre dans la partie droite du
                            ;registre no 1 le numero np.
```

```
        HRL     2,@0(16)  ;mettre dans la partie gauche du
                          ;registre no 2 le jfn du fichier.
        HRR     2,@1(16)  ;mettre dans la partie droite du
                          ;registre no 2 le numero fp.
        HRLZI   3,(PMZWR) ;acces en ecriture.
        PMAP
        ERJMP   ERPMPW
        MOVEI   3,0
        MOVEM   3,@3(16)
        POPJ    17,0

ERPMPW: MOVEI    3,1
        MOVEM   3,@3(16)
        POPJ    17,0
        END



        TITLE BMUNMP
;SUBROUTINE BMUNMP (np)
;effectuer l' "unmapping" d'une page entre un processus
;et un fichier.
;argument donne par le registre no 16.
;mot 0 : np

        SEARCH MACSYM,MONSYM
        SALL

        ENTRY   BMUNMP
BMUNMP: SETO     1,       ;mettre -1 dans le registre no 1.
        HRLI    2,.FHSLF;mettre dans la partie gauche du
                         ;registre no 2 le nom du processus
                         ;courant.
        HRR     2,@0(16);mettre dans la partie droite du
                         ;registre no 2 le numero np.
        HRRZI   3,1      ;mettre dans la partie droite du
                         ;registre no 3 le nombre de pages
        PMAP
        POPJ    17,0
        END
```

4. Tests et évaluation.

La dernière partie de ce mémoire est consacrée à une série de tests d'intégration portant sur les performances du manipulateur décrit dans les chapitres précédents.
Malheureusement, le temps nous ayant fait défaut, ces tests ne nous permettent pas, loin s'en faut, de tirer des conclusions définitives et résolument optimistes. Ils nous apportent cependant des informations non négligeables. Ainsi, nous pouvons raisonnablement admettre que, d'une manière générale, ce manipulateur semble se comporter conformément à ce que nous avons prévu.
L'idéal eût été bien sûr d'implémenter le manipulateur de séries de Poisson en utilisant notre manipulateur de blocs. Nous aurions pu alors le comparer directement avec le logiciel existant au département de mathématique (c'est-à-dire M.S., cfr 1.1.). Mais l'organisation interne de M.S. nous a empêchés de réaliser cela dans un court laps de temps. En effet, dans M.S., les opérations sur les fichiers, sur les blocs et sur les items ne sont pas dissociées en des niveaux distincts et en outre, elles comprennent des actions internes aux items (un item est un coéfficient d'une série), ce dont nous ne nous soucions pas dans notre manipulateur de blocs. En bref, les concepts de manipulation algébrique de séries (addition, multiplication..) et de manipulation des entités logiques que sont les fichiers, les blocs et les items (activation, désactivation, création...) sont confondues en un tout avec pour conséquence de compliquer assez bien l'adaptation de M.S. avec les routines proposées dans

le manuel utilisateur du chapitre précédent.
Suite à ces considérations, nous avons procédé de la façon suivante:

1. Nous avons testé de façon individuelle les modules proposés à l'utilisateur ainsi que tous les modules rédigés en MACRO (c'est-à-dire que nous avons vérifié que leurs spécifications sont correctement réalisées).

2. Ensuite, nous avons effectué quelques tests d'intégration portant essentiellement sur les accès aux items d'un bloc: ces accès étant soit des lectures, soit des écritures, soit des modifications effectuées de façon séquentielle ou aléatoire.

3. Enfin, en dernier ressort, nous avons tiré quelques conclusions quant aux différentes options possibles pour améliorer ce manipulateur et pour en faire une évaluation plus complète.

Nous proposons de ne pas nous attarder sur les tests individuels, mais d'insister plutôt sur les tests d'intégration et sur les prolongements possibles de ce mémoire.
Le premier type de test que nous avons élaboré est un accès séquentiel aux items d'un bloc que ce soit en lecture, en création ou en modification. Dans ce contexte, les temps de lecture et de création de tous les items d'un bloc se révèlent

sensiblement égaux (ceci pouvant s'expliquer par le fait qu'en lecture, les entrées/sorties s'effectuent uniquement dans le sens "fichier-mémoire virtuelle" et que dans l'autre cas, les entrées/sorties se font seulement dans l'autre sens); voici quelques exemples relatifs à la lecture et à la création :
(précisons que nous réservons chaque fois 200 pages de la mémoire virtuelle pour y stocker nos données.)
pour la lecture (et la création), les temps respectifs utilisés sont:

- **1.788" [1.627"]** pour 5000 items de 5 mots [soit 50 pages],

- **3.619" [3.248"]** pour 10000 items de 5 mots [soit 99 pages],

- **7.382" [6.510"]** pour 20000 items de 5 mots [soit 197 pages],

- **9.308" [8.484"]** pour 25000 items de 5 mots [soit 246 pages],

- **11.135" [10.305"]** pour 30000 items de 5 mots [soit 295 pages],

- **18.871" [17.364"]** pour 50000 items de 5 mots [soit 491 pages].

Quant à la modification de tous les items d'un bloc, dans les mêmes conditions énoncées ci-dessus, les temps CPU respectifs s'élèvent à **2.589" ,5.113" ,10.617" ,13.653" ,16.866" ,30.447"** . Nous constatons que ces temps croissent plus ou moins proportionnellement avec la taille des blocs. Mais, dès que la dimension du bloc dépasse la place réservée en mémoire virtuelle (dans nos tests, 200 pages), nous pouvons relever une faible augmentation due au fait que le système commence à paginer. Cette variation est cependant beaucoup plus sensible lorsque nous procédons à des modifications, ainsi qu'en témoignent les derniers temps CPU notés ci-dessus. Pour ces tests, nous avons pu réaliser quelques comparaisons (en lecture) avec M.S. et dans ce cas, les résultats obtenus donnent à penser que le manipulateur travaille un peu plus rapidement.

Cependant, nous enregistrons des différences plus significatives en analysant les résultats des tests du second type. Dans les mêmes conditions, nous accédons aux items d'un bloc, mais cette fois le numéro logique de l'item accédé est un entier naturel produit par un générateur aléatoire (la fonction FORTRAN-20 appelée RAN). Ainsi, nous placons notre manipulateur dans la plus mauvaise situation qui soit et bien sûr, il commence à paginer énormément. Mais ce qui est révélateur, c'est que le logiciel M.S. placé dans une situation plus favorable (les items appartiennent à un arbre balancé et sont référencés suivant leur localisation dans cet arbre) ne se comporte pas mieux: nous avons pu le vérifier pour des séries de 15000 termes de 5 mots où, du côté de M.S., la lecture de 1000 items prend en moyenne

0.67" alors que pour notre manipulateur, cela se chiffre à **0.32"** . D'ailleurs, voici un échantillon des résultats obtenus en faisant N accès aléatoires dans un bloc de N items (le premier temps correspond à la lecture et le second temps à des modifications):

- **5.812"** **[8.083"]** pour 15000 items de 5 mots [soit 148 pages],

- **7.807"** **[11.070"]** pour 20000 items de 5 mots [soit 197 pages],

- **109.993"** **[213.068"]** pour 25000 items de 5 mots [soit 246 pages],

- **215.828"** **[431.253"]** pour 30000 items de 5 mots [soit 295 pages],

- **703.760"** **[1313.305"]** pour 50000 items de 5 mots [soit 491 pages].

Nous remarquons que lorsque la dimension du bloc dépasse la capacité de la place réservée [200 pages], les temps augmentent de façon prohibitive : il suffit d'examiner la variation lorsque nous passons de 20000 items à 25000 items: le temps d'exécution croît d'un facteur proche de 20. En effet, à ce moment, toutes les pages virtuelles renseignées dans la carte sont occupées et pourtant, il y a encore des demandes d'allocation et le système

se met à paginer. Une fois ce seuil dépassé, les temps continuent à grimper, mais le facteur de croissance est nettement plus bas. Par contre, ce qui est caractéristique dans ces chiffres, c'est que les temps correspondant aux modifications des items croissent beaucoup plus vite que les temps de lecture de ces mêmes items. Cela doit provenir du fait suivant qui ne fait que s'amplifier quand la taille des données dépasse la place mémoire réservée à cet effet:

lorsqu'une page de données perd la page virtuelle qu'elle occupait (c'est le cas où la mémoire virtuelle est saturée), cette dernière est recopiée sur fichier si elle a subi des modifications. En clair, cela signifie que lorsque le système pagine et que les opérations effectuées sur les items sont des modifications, à chaque allocation de page virtuelle correspondent deux entrées/sorties (l'une dans le sens "fichier-mémoire virtuelle" et l'autre dans le sens opposé) pour une seule entrée/sortie dans le cas où les items sont seulement accédés en lecture. Une dernière remarque est à formuler: nous pourrions être étonnés du fait que les lectures d'items prennent un peu plus de temps que les créations d'items alors que ces opérations engendrent une seule entrée/sortie mais dans des sens différents. C'est précisément cette distinction qui explique les écarts ainsi constatés: en effet, toute lecture s'effectue par l'intermédiaire d'un buffer qui est ensuite recopié dans la page virtuelle adéquate ce qui n'est pas le cas pour une opération d'écriture. Les légères différences enregistrées proviennent donc de cette recopie exigée par la structure interne de l'appel moniteur

utilisé pour les entrées/sorties.

Enfin, le dernier type de test auquel nous nous sommes intéressés envisage le cas où l'application de l'utilisateur manipule deux fichiers "multi-blocs":

- le premier contenant 20 blocs, constitués eux-mêmes de 2000 items de 5 mots,

- le second étant formé de 10 blocs contenant chacun 1000 items de 5 mots.

Ainsi, la réunion de ces deux fichiers regroupe 50000 items. Notre propos est ici de comparer les temps de création ou de lecture d'items par rapport aux temps rencontrés précédemment pour des opérations identiques portant alors sur un seul fichier de 50000 items. Le temps de création de ces deux fichiers s'élève à 21.300" contre 17.364" pour un fichier unique de 50000 items. Cette différence s'explique aisément par le fait qu'il y a 30 désactivations de blocs à effectuer dans le premier cas (30 pages "directory" à recopier sur fichier), alors que dans le second, il y en a une seule (une seule page "directory" à éjecter , puisqu'il n'y a qu'un seul bloc). Pour la lecture, nous avons testé le cycle suivant:quatre accès aléatoires et consécutifs à 4 items d'un bloc du premier fichier choisi de facon aléatoire et ensuite un accès aléatoire à un item d'un bloc du second fichier lui aussi choisi de facon aléatoire. Pour ce test, la durée de l'exécution est de 873.918", temps supérieur au temps de lecture aléatoire des 50000 items d'un fichier unique (

703.60"). A quoi peut-on attribuer cette différence ? Certainement à un engorgement plus important de la zone de la mémoire virtuelle réservée aux données de l'utilisateur. En effet, dans le cas d'un fichier unique de 50000 items, deux pages virtuelles sont assignées à deux pages "directory", alors que dans le cas présent , 60 pages de cette zone sont dès le début des opérations "gelées" au profit de 60 pages "directory" correspondant à l'activation des 30 blocs des deux fichiers. De plus, une page virtuelle ne pouvant contenir des items provenant de deux blocs différents le phénomène de fragmentation de la mémoire virtuelle s'accroît d'autant plus que le nombre de blocs simultanément actifs est élevé. Ainsi, le fichier unique de 50000 items nécessite 491 pages de données alors que nos deux fichiers comprenant globalement le même nombre d'items occupent ensemble 500 pages de données. Peut-être les priorités des blocs sont-elles mal posées et ont-elles une influence importante sur les temps d'exécution de ces accès. En tout cas, rien n'est moins sûr et nous pouvons ainsi constater qu'à ce niveau, l'analyse complète de notre manipulateur de blocs est loin d'être terminée et qu'elle exigerait une étude qui, faute de temps, ne peut être envisagée dans le cadre de ce mémoire.

En guise de conclusion à ce travail, nous donnons une liste (qui est loin d'être exhaustive) de problèmes qui mériteraient d'être traités dans le cadre d'une amélioration globale de la version actuelle de ce manipulateur de blocs de données.

Il serait intéressant d'étudier les performances du manipulateur en fonction des paramètres d'initialisation que sont la taille de

la mémoire réservée pour le stockage des données et le plafond maximal accordé à la priorité des pages. De même, pour une application déterminée, l'assignation d'une priorité adéquate à un bloc pourrait-elle contribuer à une optimisation du système: nous pensons ici à l'inévitable phénomène de pagination. Pour réaliser cela, des simulations d'applications concrètes pourraient certainement nous apporter quantité d'informations quant au comportement du système en fonction des priorités des pages et des blocs. En fait, nous nous heurtons ici à un problème rencontré par beaucoup de concepteurs de systèmes d'exploitation: la connaissance à l'avance de la facon dont les pages seront référencées dans le cadre d'une application donnée, ce problème se trouvant pour nous au niveau des références aux blocs et à leurs items.

Une deuxième direction dans laquelle nous pourrions nous engager serait d'étoffer l'ensemble des traitements de base proposés à l'utilisateur. Nous pensons notamment à la mise au point d'une routine de terminaison qui se substituerait au "stop" du programme principal de l'utilisateur. Son rôle serait de vérifier que tous les blocs activés en cours d'exécution ont été désactivés et dans le cas contraire, de procéder à la désactivation de ces blocs. En outre, cette routine émettrait une série de statistiques quant au comportement général du système et pourrait peut-être apporter des renseignements précieux quant à la facon dont ses blocs et leurs items sont référencés durant l'exécution du programme.

Le lecteur peut donc se rendre compte aisément qu'il reste encore

beaucoup de choses à améliorer. Néanmoins, nous espérons avoir contribué d'une manière ou d'une autre à une meilleure connaissance de ce sujet.

A l'intention des personnes désireuses d'obtenir des informations complémentaires sur le sujet abordé dans ce mémoire, nous signalons ci-dessous une listes de références susceptibles de les intéresser:

- WILLIAM H.JEFFERYS:

  * "A Precompiler for the Formula Manipulation System Trigman."
    Celestial Mechanics 6 (1972) 117-124.

  * "A Fortran-based List Processor for Poisson Series."
    Celestial Mechanics 2 (1970) 474-480. ,

- DAVID EVANS & ANDRIES VAN DAM.
  "Data Structure Programming System." Information Processing 68 - North-Holland Publishing Company - Amsterdam (1969).

- ARNOLD ROM.

  * "Mechanized Algebraic Operations."
    Celestial Mechanics 1 (1970) 301-319.

  * "Echeloned Series Processor."
    Celestial Mechanics 3 (1971) 331-345.

- ROGER A.BROUCKE.

"A Fortran-4 System for the Manipulation of Symbolic Poisson Series with Applications to celestial Mechanics."

Iasom Tr 80-3 January 1980.

BIBLIOGRAPHIE.

- J.HENRARD.
"Poisson Series Processor" (A User's Manual)
Publication du Département de Mathématique. Report 77/4.

- PH.DONTAINE.
"A Poisson Series Processor on a Dec-20/60 Computer".
Logiciel implémenté au département de mathématique.

- D.BARTON.
"A Scheme for manipulative Algebra on a Computer."
The Computer Journal volume 9 number 4 february 1967.

- D.BARTON & S.R.BOURNE & J.R.HORTON.
"The Structure of the Cambridge Algebra System."
The Computer Journal volume 13 number 3 august 1970.

- D.BARTON & S.R.BOURNET & J.P.FITCH.
"An Algebra System."
The computer Journal volume 13 number 1 february 1970.

- A.VAN DAM & FR.TOMPA.
"Software Data Pasing and Segmentation for complex Systems."
Information Processing Letters 1 (1972) 80-86.

North-Holland Publishing Company.

- B.RANDELL & C.J.KUEHNER.

"Dynamic Storage Allocation Systems."

The computer Journal volume 11 number 5 may 1968.

- JANE G.JODEIT.

"Storage Organization in Programming Systems."

The Computer Journal volume 11 number 11 november 1968.

- RICHARD J.FATEMAN.

"On the Multiplication of Poisson Series."

Celestial Mechanics 10 (1974) 243-247.